NASA Contractor Report-187556

Advanced Information Processing System: Inter-Computer Communication Services

Laura Burkhardt
Tom Masotto
J. Terry Sims
Roy Whittredge
Linda Alger

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-18565
September 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

NASA Contractor Report-187556

Advanced Information Processing System: Inter-Computer Communication Services

Laura Burkhardt
Tom Masotto
J. Terry Sims
Roy Whittredge
Linda Alger

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-18565
September 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

The purpose of this report is to document the functional requirements and detailed specifications for the Inter-Computer Communication Services (ICCS) of the Advanced Information Processing System (AIPS). This introductory section is provided to outline the overall architecture and functional requirements of the AIPS system and to present a overview of the Inter-Computer Communication Services. Section 1.1 gives an overview of the AIPS architecture as well as a brief description of the AIPS inter-computer network architecture; Section 1.2 provides an introduction to the AIPS system software; Section 1.3 provides the guarantees of the Inter-Computer Communication Services; and Section 1.4 describes the Inter-Computer Communication Services as a seven layered International Standards Organization (ISO) model. Sections 2 through 6 describe the Inter-Computer Communication Services functional requirements, functional design and detailed specifications. Each of these sections describes one of the 'Layers' of the Inter-Computer Communication Services. Section 7 concludes with a summary of results and suggestions for future work in this area.

1.1 AIPS Architecture

The Advanced Information Processing System is designed to provide a fault- and damage-tolerant data processing architecture which can serve as the core avionics system for many of the aerospace vehicles being researched and developed by NASA. These vehicles include manned and unmanned space vehicles and platforms, deep space probes, commercial transports, and tactical military aircraft.

AIPS is a multicomputer architecture composed of hardware and software 'building blocks' that can be configured to meet a broad range of application requirements. The hardware building blocks are fault-tolerant, general purpose computers (GPCs), fault- and damage-tolerant inter-computer (IC) and input/output (I/O) networks, and interfaces between the networks and the general purpose computers. The software building blocks are the major software functions: local system services, input/output system services, inter-computer communication services, and the system manager. This software provides the services necessary in a traditional real-time computer, such as task scheduling and dispatching, and communication with sensors and actuators. The software also supplies the redundancy management services necessary in a redundant computer and the services necessary in a distributed system such as inter-function communication across processing sites, management of distributed redundancy, management of networks, and migration of functions between processing sites.

The AIPS hardware consists of a number of computers which may be physically dispersed throughout a vehicle. These dispersed computers are linked together by a reliable, damage-tolerant data communication pathway called the IC network, or IC bus. (Since the hardware implementation is a circuit-switched network which appears to the communication software and the receiving and transmitting devices as a conventional bus, the terms 'network' and

'bus' are used interchangeably throughout this document.) A computer at any particular processing site may also have access to varying numbers and types of I/O buses, which are separate from the IC bus. The I/O buses may be global, regional or local in nature. I/O devices on the global I/O bus are available to all, or at least a majority, of the AIPS computers. Regional buses connect I/O devices in a given region to the processing sites located in their vicinity. Local buses connect a computer to the I/O devices dedicated to that computer. Additionally, I/O devices may be connected directly to the internal bus of a processor and accessed as though the I/O devices reside in the computer memory (memory mapped I/O). Both the I/O buses and the IC bus are time-division multiple-access contention buses. Figure 1-1 shows the laboratory engineering model for a distributed AIPS configuration. This distributed AIPS configuration includes all the hardware and software building blocks mentioned earlier and was conceived and built to demonstrate the feasibility of the AIPS architecture.

The laboratory configuration of the distributed AIPS system shown in Figure 1-1 consists of four processing sites: three of the GPCs are triplex FTPs, while the fourth GPC is a simplex. Processing site 4 with its 15 node I/O network forms the centralized AIPS configuration, which is a subset of the distributed AIPS configuration. The interfaces between the GPCs and the IC network, shown in Figure 1-1, are called Inter-Computer Interface Sequencers (ICIS). The interfaces between the FTP and the I/O network are called Input/Output Sequencers (IOS). The redundant FTPs are built such that they can be physically dispersed for damage tolerance; each of the redundant channels of a FTP can be as far as 5 meters from other channels of the same FTP.

The GPCs communicate with each other over the Inter-Computer Network, in which the circuit-switching nodes have been configured into redundant virtual buses. Each redundant bus is referred to as a layer (not to be confused with the ISO layers); these layers are totally independent and are not cross-strapped to each other. Each layer contains a circuit-switched node for each processing site; thus every processing site is serviced by three nodes of the IC network. GPCs are designed to receive data on all three layers, but the capability of a GPC to transmit on the network depends on the GPC redundancy level. Triplex FTPs can transmit on all three layers, duplex FTPs on only two of the three layers, and simplex processors on only a single layer. In duplex and triplex FTPs, a given processor can transmit on only one network layer. Thus malicious behavior of a processor can disrupt only one layer.

The IC network and the interfaces into the network are designed in strict accordance with fault-tolerant systems theory. An arbitrary random hardware fault, including Byzantine faults, anywhere in the system can not disrupt communication between triplex FTPs. In other words, the triplex IC network, in conjunction with the ICIS, provides error-masking capability for communication between two triplex computers. The IC network architecture is described in more detail in the following subsection.

AIPS ENGINEERING MODEL CONFIGURATION

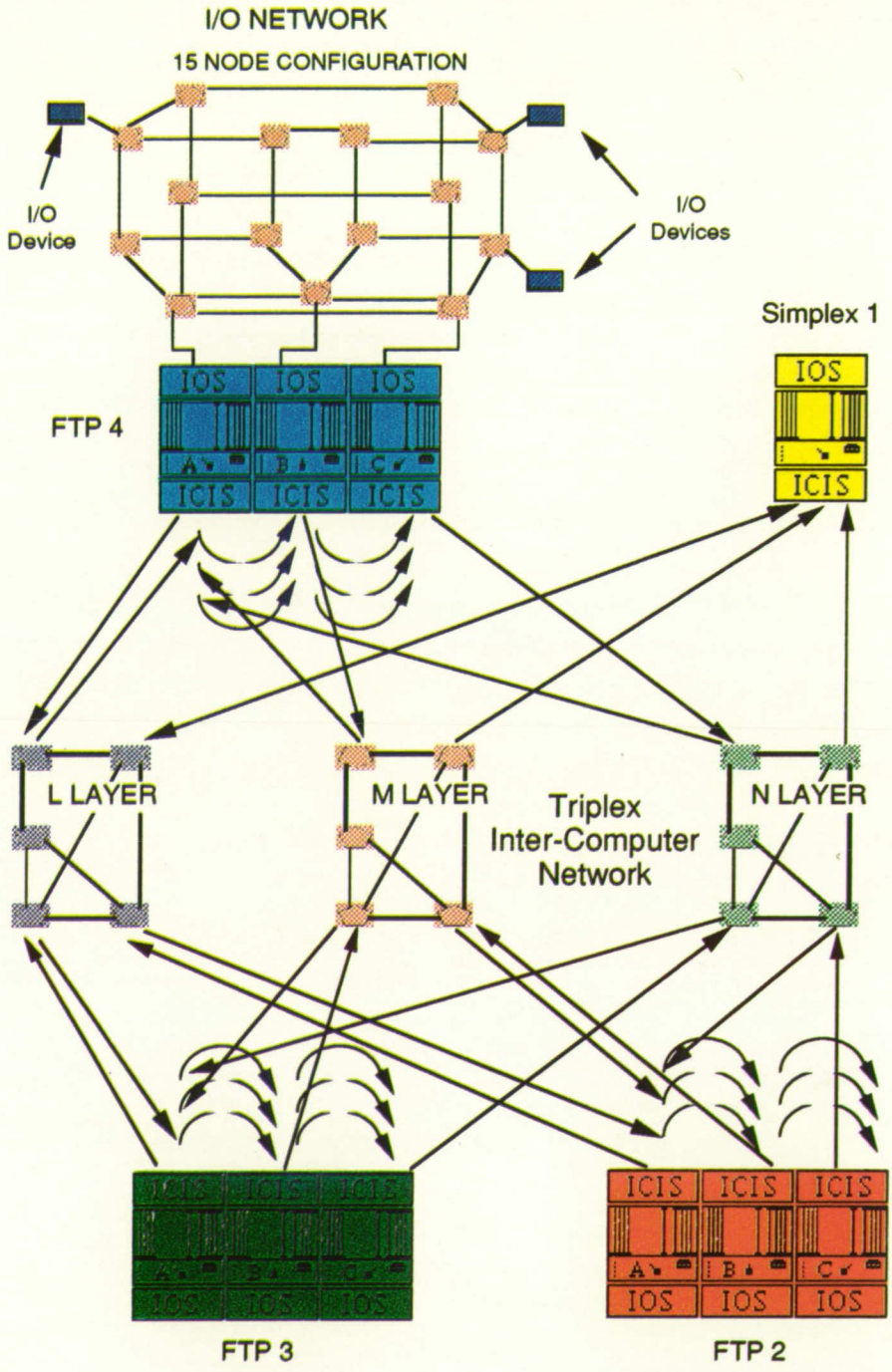


Figure 1-1. AIPS Engineering Model Distributed Configuration

1.1.1 AIPS Inter-Computer Network Overview

Communication among the distributed computers of the Advanced Information Processing System on the IC network is enabled by circuit switched nodes. These nodes are identical to those used on the I/O network for communication between processors and sensors and actuators. Each node has five identical ports and can interface with other nodes, GPCs and I/O subscribers (displays, etc.) through these ports. A detailed specification of the CSDL node is contained in Appendix A.

The IC network for the distributed AIPS engineering model configuration consists of three layers of a circuit switched network. Each layer consists of five nodes: one node for each of the four sites and one spare node.

The three layers of the IC network are totally independent and are not cross-strapped to each other. The initial no-fault configurations of the three layers are identical. However, after a link failure in one layer the virtual bus configuration of that layer would change as the network is reconfigured around the failed link. The other two layers do not have to be reconfigured to make their virtual bus path identical to the third one.

The fault detection, isolation, and reconfiguration of the IC network is the responsibility of the IC Network Manager. Nodes keep track of any transmission errors which are protocol related and inform the Manager of these errors when queried by the Manager. These error data can be analyzed by the Network Manager to determine the source of transient faults on the network. The nodes also respond to status queries with the status of the node and the ports. Other than these functions, the nodes are totally passive circuit switching devices. The common control circuits in a node monitor messages coming in on all five ports whether that port is enabled or not. This procedure is necessary for the initial growth of the network. It is also necessary to monitor all ports so that the Network Manager can respond to certain kinds of failures where the established paths have been disrupted by a malicious failure. The controller decodes the message to determine if it is a valid message and if it is intended for that node. If the message is valid and intended for the particular node, the node responds to the message. Messages sent to nodes include requests for status and reconfiguration commands. The Network Manager requests status as an input to its network monitoring task. The reconfiguration messages establish or change the port enable status. Reconfiguration commands must be preceded by an encoded node address. Nodes do not respond to messages which are not preceded by valid addresses. This validity test is done in order to prevent a GPC with random hardware failure from reconfiguring the network. The reconfiguration commands are addressed to individual nodes although they are heard by all nodes.

1.2 AIPS System Software

The AIPS system software, as well as the hardware, has been designed to provide a virtual machine architecture that hides hardware redundancy, hardware faults, multiplicity of

resources, and distributed system characteristics from the applications programmer. Section 1.2.1 discusses the approach used for the AIPS system software design. Section 1.2.2 is a high level description of the system services that are provided for AIPS users.

1.2.1 AIPS Software Design Approach

The approach used to design the AIPS system software is part of the overall AIPS system design methodology. An abbreviated form of this system design methodology is shown in Figure 1-2. This methodology began with the application requirements and eventually led to a set of architectural specifications. The architecture was then partitioned into hardware and software functional requirements. This report documents the design approach used for Inter-Computer Communication Services software and the ICIS hardware, beginning with the functional requirements and proceeding through detailed specifications.

Hardware and software for the AIPS architecture is being designed and implemented in two phases. The first phase was the centralized AIPS configuration. The centralized AIPS architecture, as shown in Figure 1-3, consists of one triplex Fault Tolerant Processor (FTP), an Input/Output network and the interfaces between the FTP and the network, the IOSes. The laboratory demonstration of the Input/Output network consists of 15 circuit-switched nodes which can be configured as multiple local I/O networks connected to the triplex GPC. The second phase is the distributed AIPS configuration, which was explained earlier and shown in Figure 1-1.

1.2.2 AIPS System Software Overview

As shown in Figure 1-4, AIPS system software provides the following AIPS System Services: local system services, I/O system services, inter-computer communication services, and system management. The system software is being developed in Ada. System services are modular and partitioned naturally according to hardware building blocks. The distributed AIPS configuration will include all the services. (At this time, all the system services with the exception of system management have been completed.) Versions of the system software for specific applications can be created by deleting unused services from this superset. The System Manager functions may reside on only one triplex FTP or may be distributed among several triplex FTPs. The other system services are replicated in each GPC. A brief description of each of the services follows.

1.2.2.1 Local System Services

The local system services provided in each GPC are GPC initialization, real-time operating system, local resource allocation, local GPC Fault Detection, Isolation, and Reconfiguration (FDIR), GPC status reporting, and local time management (see Figure 1-5).

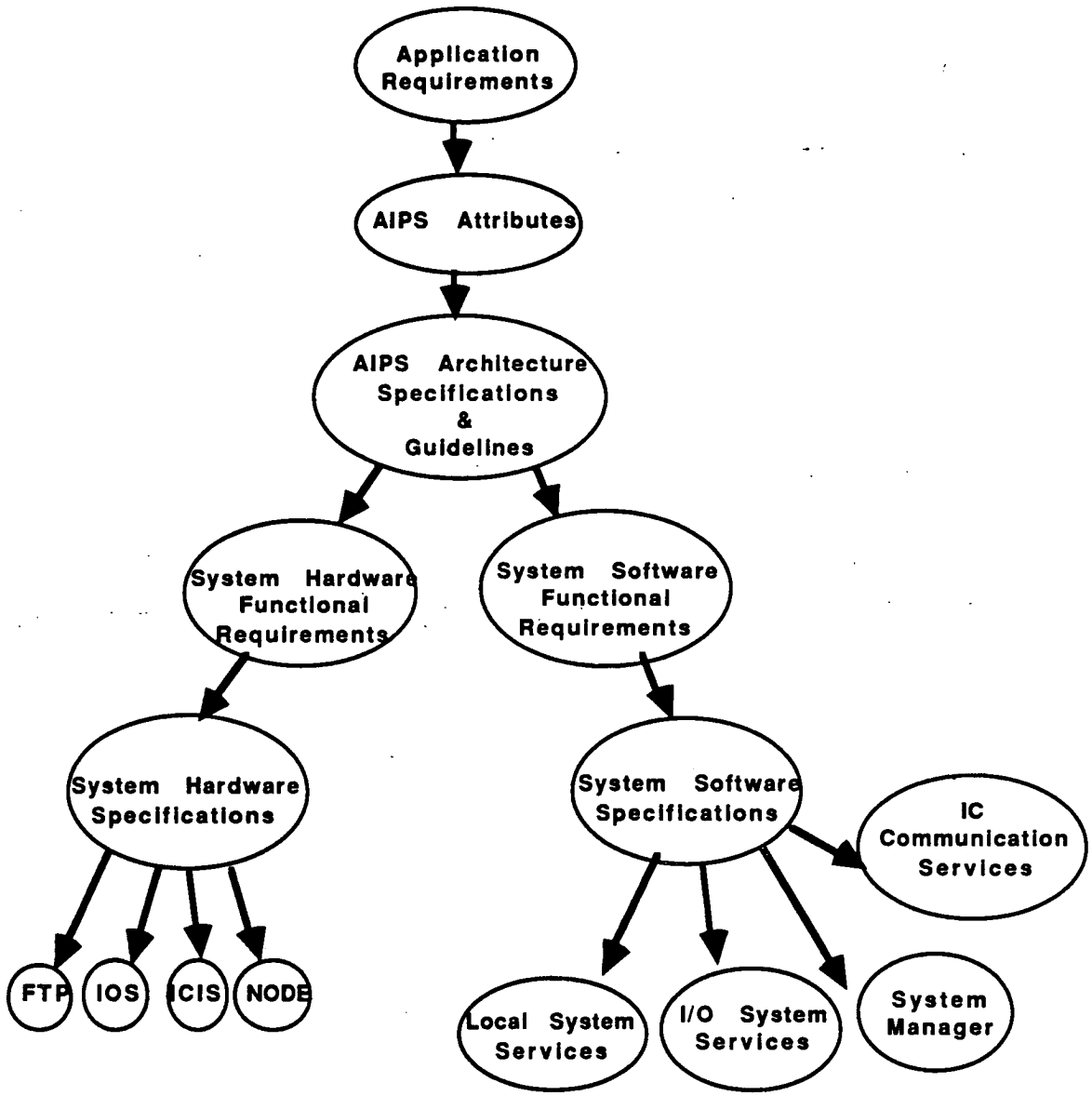


Figure 1-2. AIPS System Design Approach

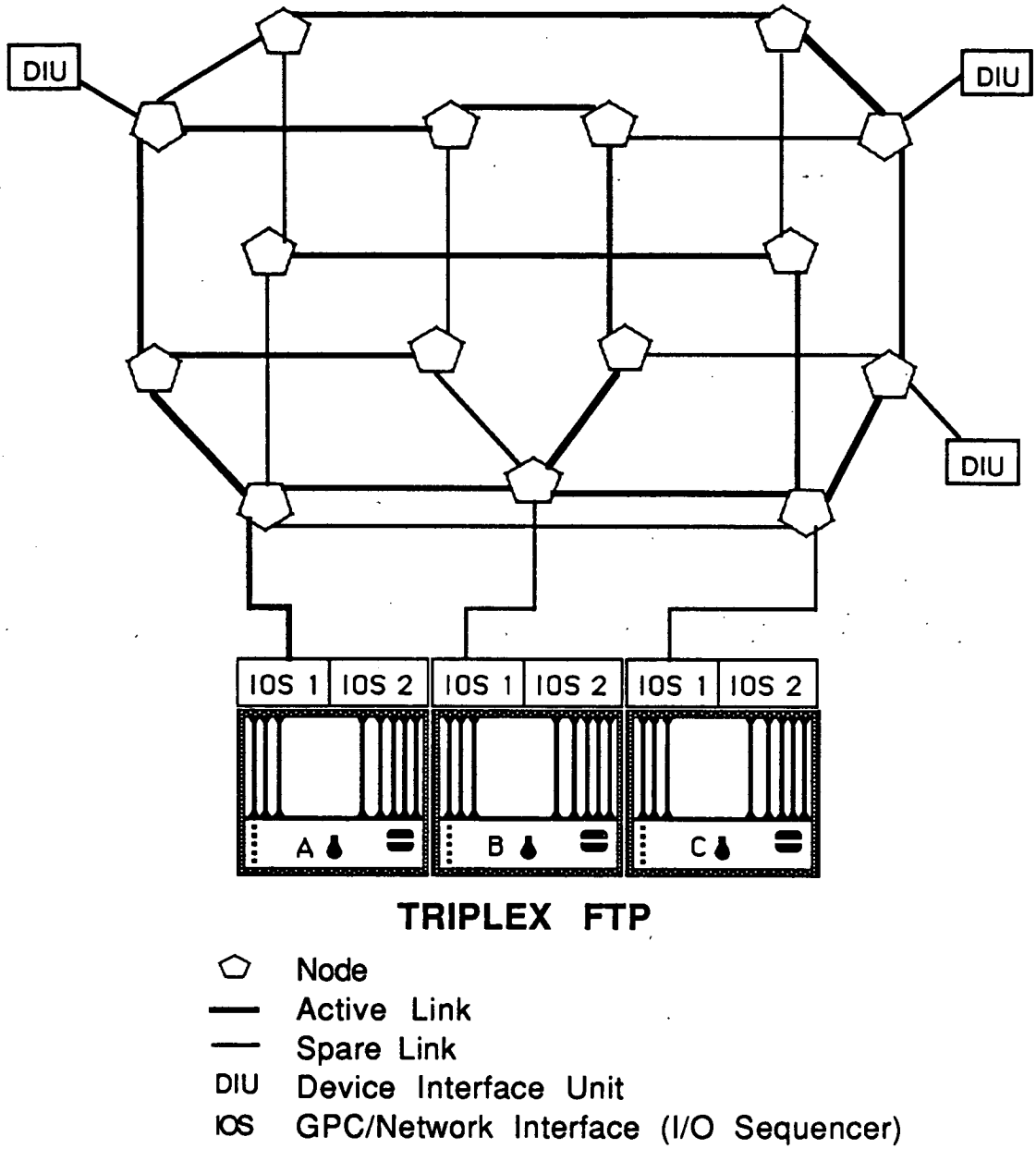


Figure 1-3. Centralized AIPS Configuration

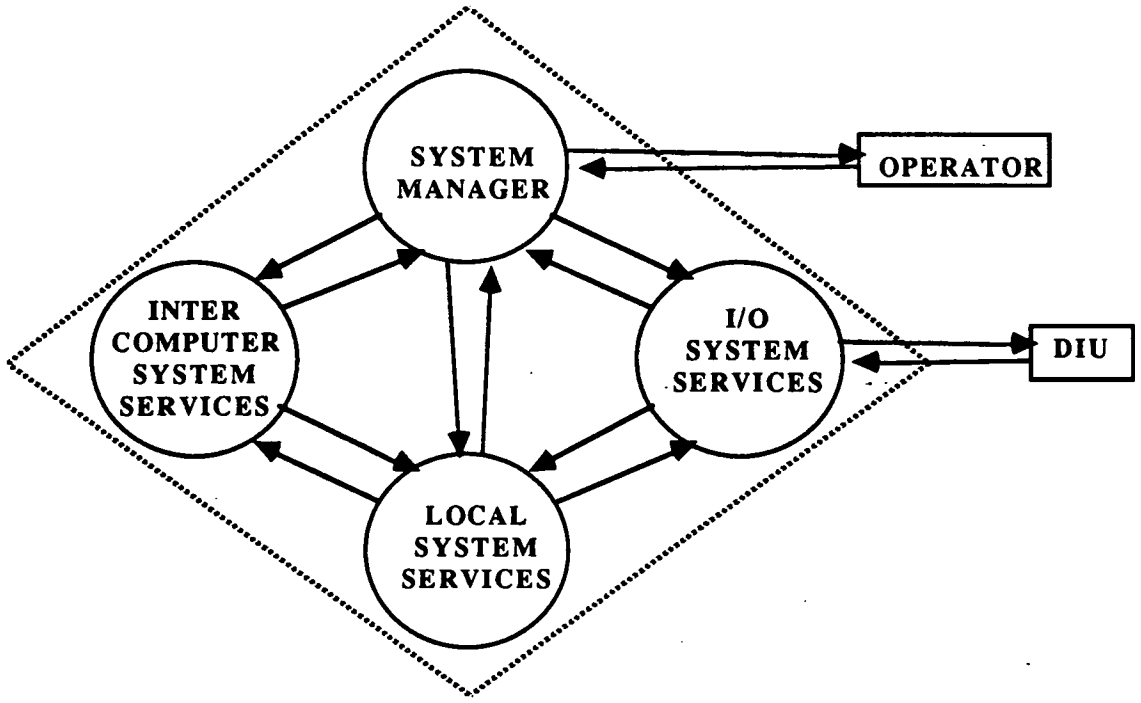


Figure 1-4. Top Level View Of System Services

The function of GPC initialization is to bring the GPC to a known and operational state from an unknown condition (cold start). GPC initialization synchronizes the Computational Processors (CPs), synchronizes the Input/Output Processors (IOPs) and resets or initializes the GPC hardware and interfaces (interval timers, real time clock, interface sequencers, DUART, etc.). It makes the hardware state of the redundant channels congruent by alignment of memory and control registers. It then activates the system baseline software that is common to every GPC.

The AIPS real-time operating system supports task execution management, including scheduling according to priority, time and event occurrence, and is responsible for task dispatching, suspension and termination. It also supports memory management, software exception handling, and intertask communication between companion processors (IOP and CP). The AIPS operating system resides on every CP and IOP in the system. It uses the vendor-supplied Ada Run Time System (RTS), and includes additional features required for the AIPS real-time distributed operating system.

The GPC resource allocator coordinates and manages any global or migratable functions that are assigned to the GPC.

The GPC status reporter collects status information from the local functions, the local GPC FDIR, the IC system services and the I/O system services. It updates its local data base and disseminates this status information to the system manager.

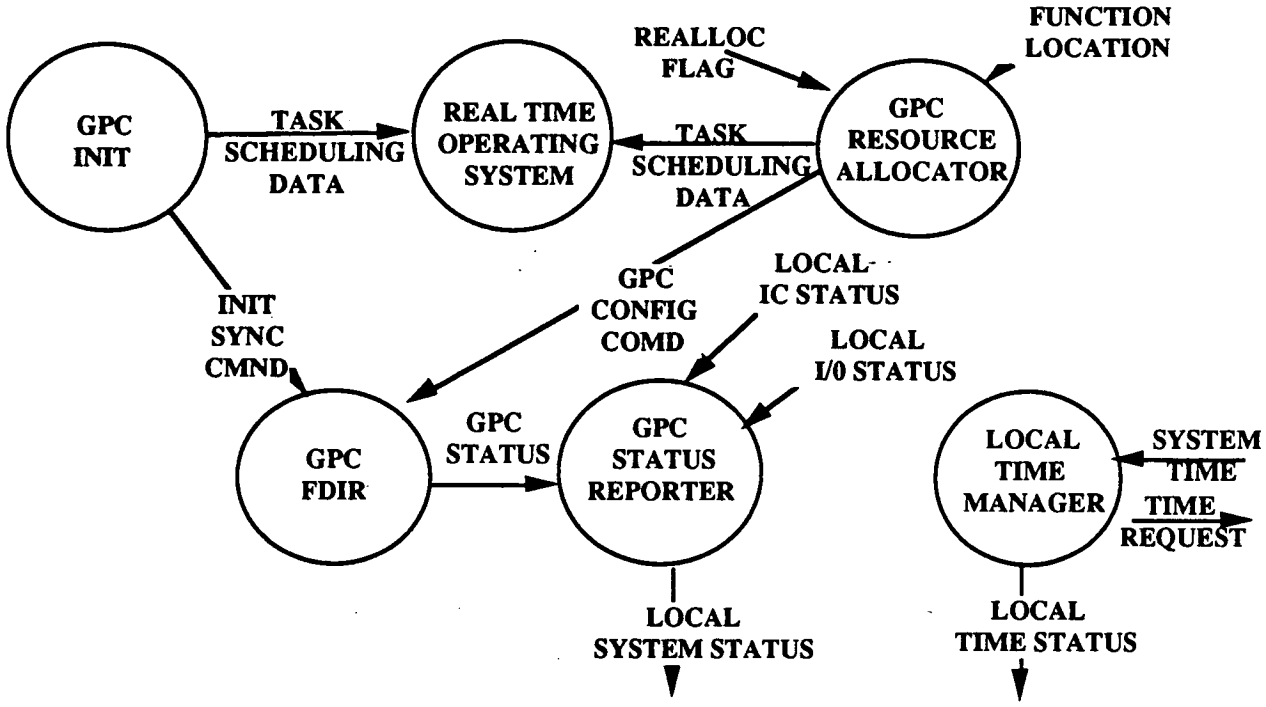


Figure 1-5. Local System Services

GPC FDIR has the responsibility for detecting and isolating hardware faults in the CPs, IOPs, and shared hardware. It is responsible for synchronizing both groups of processors in the redundant channels of the FTP and for disabling outputs of failed channel(s) through interlock hardware. After synchronization, all CPs will be executing the same machine language instruction within a bounded skew, and all IOPs will be executing the same machine language instruction within a bounded skew. GPC FDIR logs all faults and reports status to the GPC status reporter. It is responsible for the CPU hardware exception handling, for transient hardware fault detection, and for running low priority self tests to detect latent faults. This redundancy management function is transparent to the application programmer.

The local time manager works in cooperation with the system time manager to initialize the local real time and to keep it consistent with the universal time. It is also responsible for providing time services to all users.

A detailed description of Local System Services is provided in [1].

1.2.2.2 I/O System Services

The I/O system services provide efficient and reliable communication between the user and external devices (sensors and actuators). The I/O system services software is also responsible

for the fault detection, isolation and reconfiguration of the I/O network hardware and the IOSes.

I/O system services is made up of three functional modules: I/O user interface, I/O communication management and the I/O network manager (Figure 1-6).

The I/O User Interface provides a user with read/write access to I/O devices or Device Interface Units (DIUs), such that the devices appear to be memory mapped. It also gives the user the ability to group I/O transactions into chains and I/O requests, and to schedule I/O requests either as periodic tasks or on demand tasks. A detailed description of the I/O user interface is provided in [2].

The I/O Communication Manager provides the functions necessary to control the flow of data between a GPC and the various I/O networks used by the GPC. It also performs source congruency and error detection on inputs, votes all outputs, and reports communication errors to the I/O Network Manager. Additionally, it is responsible for the management of the I/O request queues. A detailed description of the I/O communication manager is provided in [2].

The I/O Network Manager is responsible for detecting and isolating hardware faults in I/O nodes, links, and interfaces and for reconfiguring the network around any failed elements. The network manager function is transparent to all application users of the network. A detailed description of the I/O Network Manager is provided in [3].

1.2.2.3 Inter-Computer Communication Services

The Inter-Computer Communication Services provide two functions: (1) inter-computer user communication services, that is, communication between functions not located in the same GPC, and (2) inter-computer network management (Figure 1-7).

The characteristics of the Inter-Computer Communication Services are described in Section 1.3. The ICCS has been designed and implemented according to the ISO model [4]. This model and the mapping of ICCS functions to the model are described in Section 1.4. Sections 2 through 6 describe the ICCS functional requirements, functional design and detailed specifications. Each of these sections describes one of layers of the Inter-Computer Communication Services.

The IC user communication service provides local and distributed inter-function communication which is transparent to the application user. It provides synchronous and asynchronous communication, performs error detection and source congruency on inputs, and records and reports IC communication errors to IC network managers. Inter-Computer communication can be done in either point to point or broadcast mode and executes in each GPC.

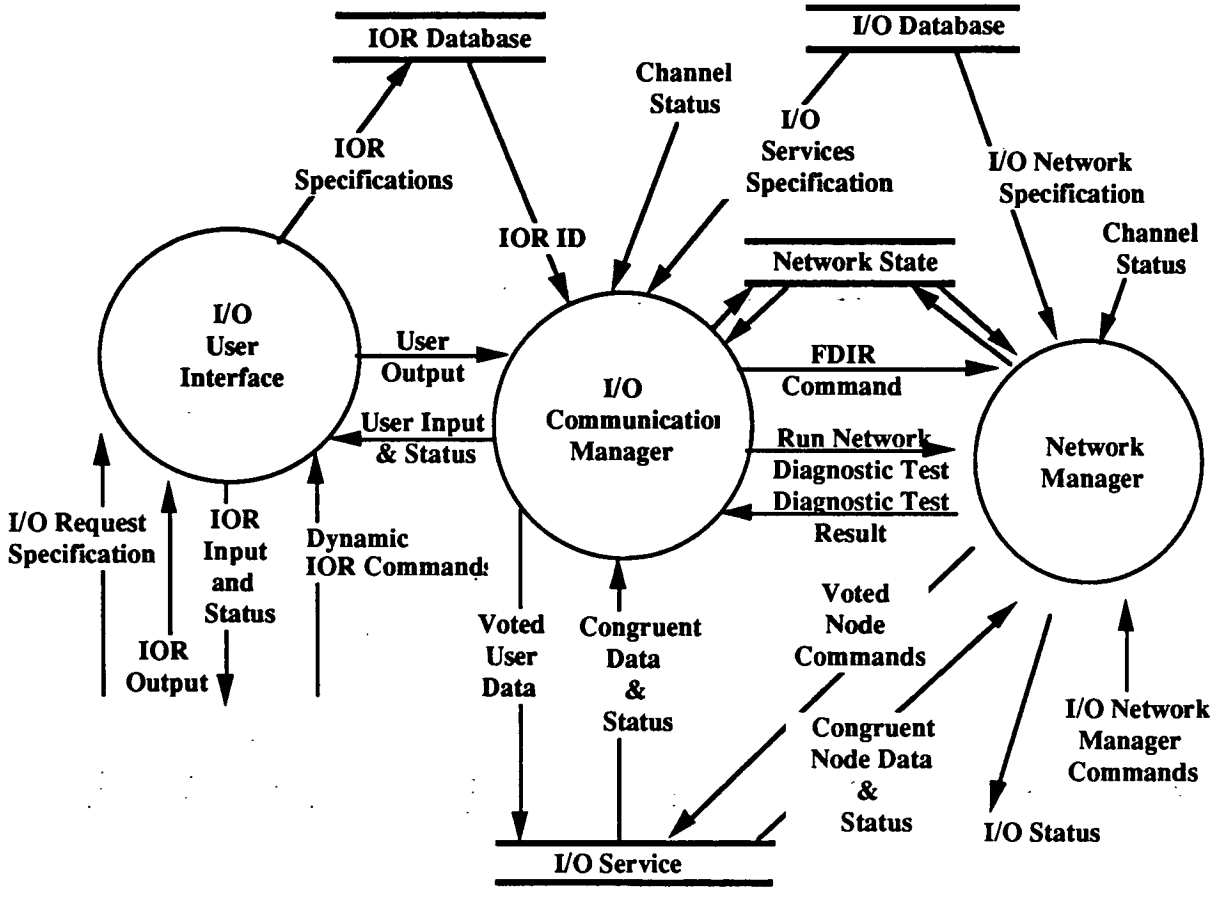


Figure 1-6. I/O System Services

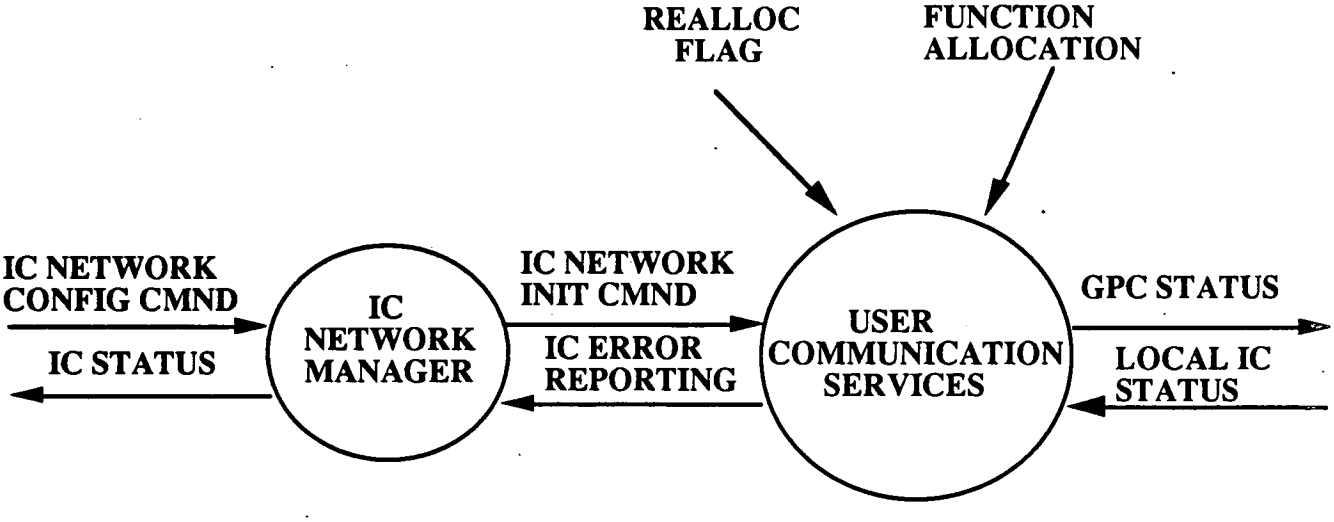


Figure 1-7. Inter-Computer System Services

The IC Network Manager is responsible for the fault detection, isolation and reconfiguration of the IC network. The AIPS distributed configuration consists of three identical, independent IC network layers which operate in parallel to dynamically mask faults in a single layer and provide reliable communication. The IC Network Manager is composed of three IC Layer Managers. There is one Layer Manager for each network layer. However, the three Layer Managers do not need to reside in the same GPC. They are responsible for detecting and isolating hardware faults in IC nodes and links and for reconfiguring their respective network layer around any failed elements. The Network Manager function is transparent to all application users of the network.

1.2.2.4 System Manager

The system manager is a collection of system level services including the applications monitor, the system resource manager, system fault detection, isolation and reconfiguration (FDIR), and the system time manager (Figure 1-8).

The applications monitor interfaces with the applications programs and the AIPS system operator. It accepts commands to migrate functions from one GPC to another, to display system status, to change the state of the system by requesting a hardware element state change, and to convey requests for desired hardware and software configurations to the system resource manager.

The system resource manager allocates migratable functions to GPCs. This involves the monitoring of the various triggers for function migration such as failure or repair of hardware components, mission phase or workload change, operator or crew requests and timed events. It reallocates functions in response to any of these events. It also designates managers for shared resources and sets up the task location data base in each GPC.

The system FDIR is responsible for the collection of status from the IC Network Layer Managers, the I/O Network Managers, and the local GPC redundancy managers. It resolves conflicting local fault isolation decisions, isolates unresolved faults, correlates transient faults, and handles processing site failures.

The system time manager, in conjunction with the local time manager on each GPC, has the job of maintaining a consistent time across all GPCs. The system time manager indicates to the local time manager when to set its value of time. It also sends a periodic signal to enable the local time manager to adjust its time to maintain consistency with an external time source such as the GPS Satellites or an internal source such as the real time clock in the GPC which hosts the system time manager software.

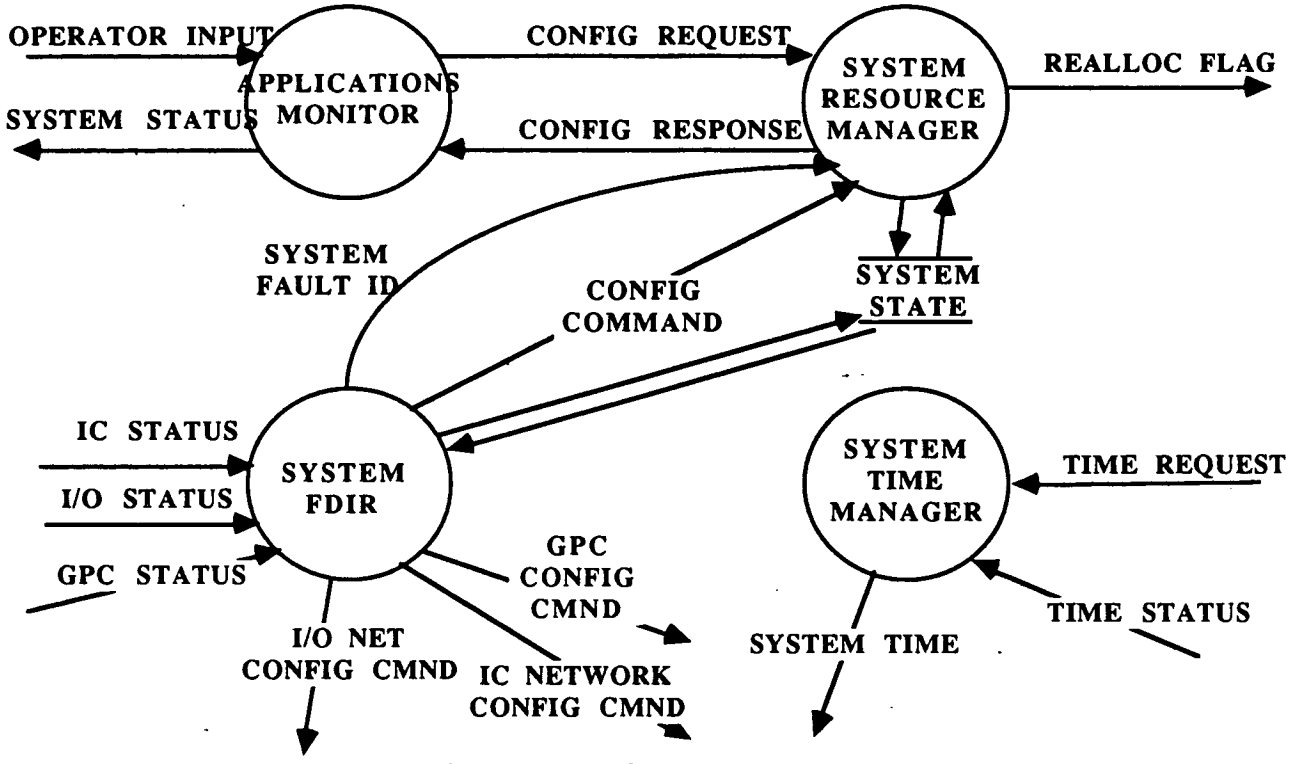


Figure 1-8. System Manager

1.3 Inter-Computer Communication Services Characteristics

The Inter-Computer Communication Services possess a number of characteristics that make it much easier for the applications programmer to use these services. These characteristics also off-load the applications programmer as well as the systems programmer from the tasks of building reliable message delivery system on top of an unreliable communications medium and ordering and timing of delivery of messages. The following is a list of terms and definitions that are used in the subsequent discussion.

1. An FTP is 'on-line' if the majority of its channels are non-faulty.
2. A simplex is 'on-line' if it is non-faulty and its transmittal layer is non-faulty.
3. The 'source' site is the transmitting site.
4. The 'sink' site(s) is the recipient site(s).
5. The sink sites of a broadcast are all of the on-line sites except for the source site.
6. The IC Network is 'on-line' if the majority of its layers are non-faulty.
7. A fault in this context is a random hardware fault or an operational fault that affects only a single fault containment region.

There are two types of communication supported by the ICCS, the IC Network, and ICISes: point-to-point communication and broadcast communication. The characteristics of both types of communication are the following.

In a fully operational state all triplex FTPs can communicate with each other, even in the presence of an arbitrary malicious fault.

The following are the characteristics of AIPS for point to point (site to site) communication.

1. **Messages sent by an on-line source site are correctly delivered to the on-line sink site.**

2. **An on-line sink site receives messages in the order sent by the on-line source site.**

3. **A task residing on an on-line sink site receives messages in the order sent by tasks residing on one other on-line source site.**

Figure 1-9 is a diagram of message flow for point to point communication. In the diagram, Task A residing on FTP 2 sends two message to Task B residing on FTP 4. All four sites are 'on-line.' Message X is sent first and message Y is sent second. The messages are received by Task B in the order they have been sent.

The following are the AIPS characteristics for broadcast or site to sites communication.

1. **Broadcasts sent by an on-line source site are correctly delivered to all other on-line sites.**

2. **On-line sink sites receive messages in the order sent by the broadcasting sites.**

3. **On-line sink sites receive broadcast messages in identical order.**

4. **A task residing on an on-line sink site receives messages in the order sent by tasks residing on the on-line source sites.**

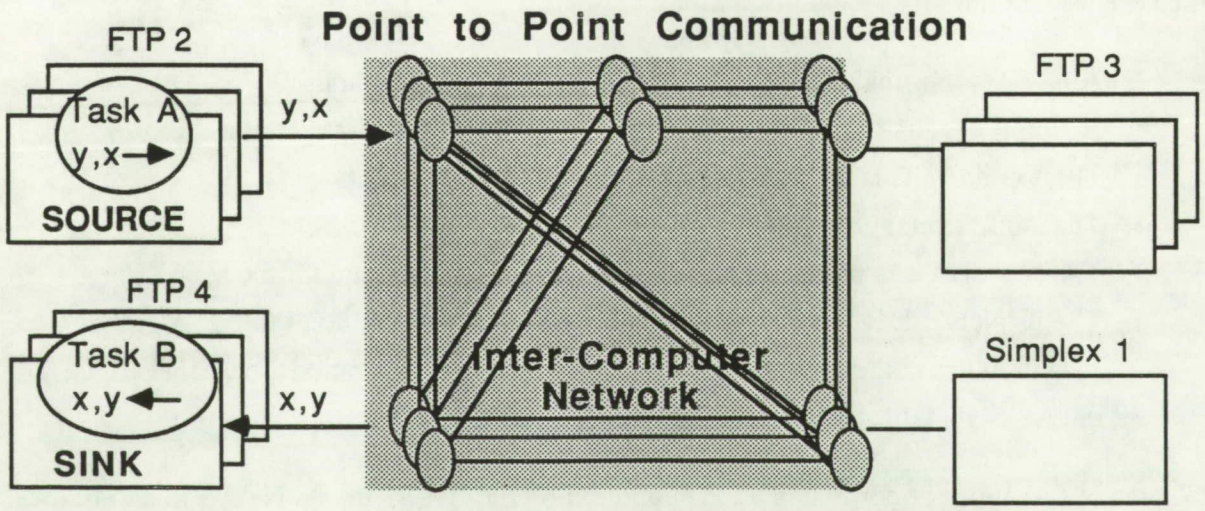


Figure 1-9. Point To Point Communication Message Flow

Figure 1-10 is a diagram of message flow for broadcast communication. In the diagram, Task A residing on FTP 4 sources the first two broadcast messages, B1 and B2. The recipient is Task B, residing on all other sites. Another task, Task C (not shown on the diagram), residing on FTP 3 then broadcasts the B3 message to a different recipient task, Task D (not shown on the diagram), on all other sites. All four sites are 'on-line.' Task B on sites 1, 2, and 3 receives the first two message in the order sent. Task D on sites 1, 2, and 4 then receive message B3.

1.4 Inter-Computer Communication Services ISO Model

The ICCS was designed and implemented in a manner which is consistent with the ISO model. This model is the standard for communication between data processing machines. The architectural framework for the ISO is a hierarchical series of seven functional levels or layers [4]. Different layers relate to different types of functions and services. Figure 1-11 is a diagram of the relationship between the ISO layers and the hardware or software tasks of the ICCS.

At the lowest level are the Physical Layer and the Data Link Layer. The Physical Layer is concerned with the electrical connection between the data machine and the communication circuit. The Data Link Layer is concerned with how blocks of data are sent over the physical link. These two layers are provided by the ICIS hardware. Section 6 is a detailed description of the functional requirements and design of the ICIS.

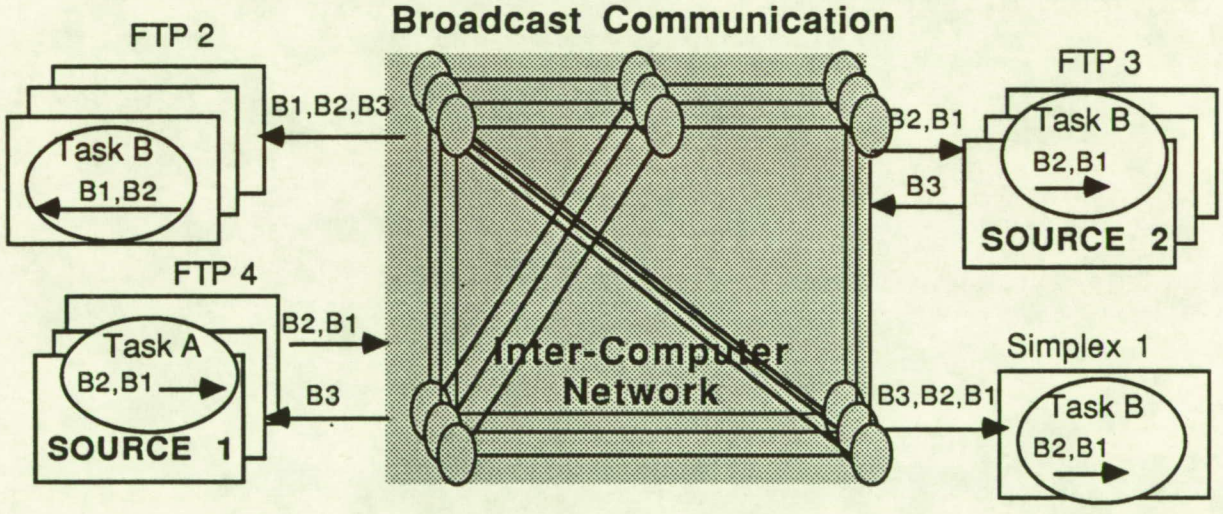


Figure 1-10. Broadcast Communication Message Flow

The next level is the Network Layer, which is concerned with setting up a virtual circuit spanning one or more physical links. This layer is provided by the IC Network Manager Task, which is responsible for the initial growth of the IC Network and the fault detection, identification and reconfiguration of the redundant links and nodes of the network. Section 5

is a detailed description of the functional requirements and software specifications of the IC Network Manager. These lower three ISO layers represent a common network which many machines might share independently of one another.

The next layer, the Transport Layer is concerned with reliable end-to-end control of the transmission between two user processes. The functions of this layer are provided by the Message Send Receive Task, the ICIS Redundancy Management Task and the User Services. The Message Send Receive Task is responsible for providing transmit and receive functions in either point to point or broadcast mode. The ICIS Redundancy Management Task is responsible for providing redundant processors with a congruent representation of received data and for detecting and isolating ICIS faults. The User Services provide a standard interface to the user. Section 4 is a detailed description of the functional requirements and software specifications of the Transport Layer.

Above the Transport Layer is the Session Layer, whose function is to begin a session between two users, maintain a dialogue according to an established protocol, and then terminate the session. This function is provided by the Synchronous Communication Manager. (Applications could also write their own Session Layer functions). Section 3 is a detailed description of the functional requirements and software specifications of the Synchronous Communication Manager.

A brief description of the highest layers, the Presentation and the Process Layers, is provided in the following Section. These layers are responsible for the manipulation of data that has been transmitted by the lower layers.

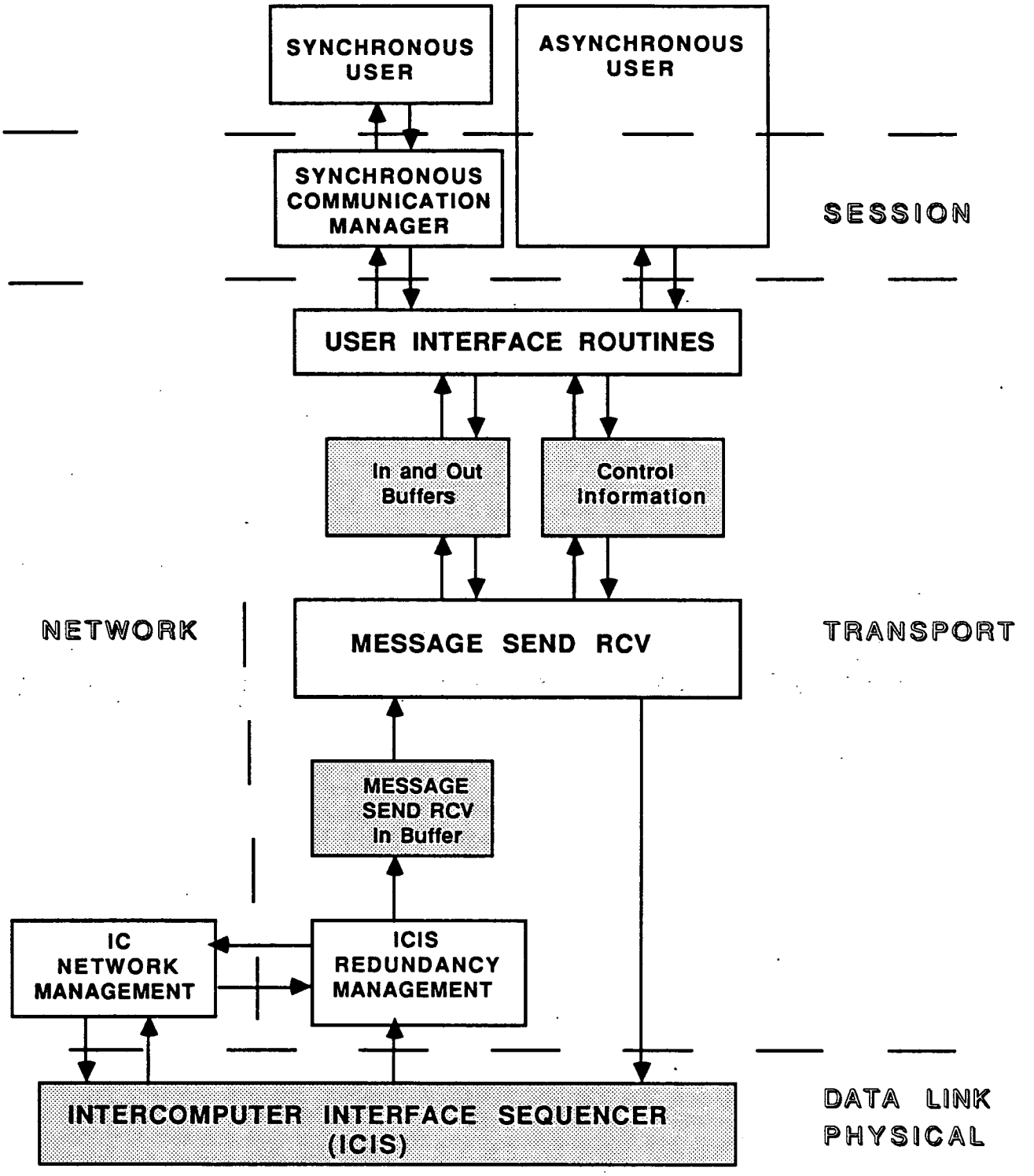


Figure 1-11. Inter-Computer Communication Services ISO Layer Components

2.0 THE PRESENTATION AND PROCESS LAYERS

The Presentation Layer and the Process Layer are the two outermost layers of the seven standard layers that the ISO has defined for distributed processing. These layers are concerned with the manipulation of data that has been transmitted by the lower layers. The Inter-Computer Communication Services provided for AIPS do not include any functions in these two layers, but a brief description is given here for completeness [4].

The Presentation Layer contains functions relating to the character set and data codes which are used and to the way data is displayed on a screen or printer. For example, when communication is occurring between devices with different character sets, character conversion needs to be performed. In order to save transmission costs, it may be desirable to compact a character stream into a smaller bit stream. For security reasons data encryption and decryption may be required. In order to have a character stream appear in an attractive and readable format on a display device or printer, control characters need to be inserted at appropriate places. Conversely, a user-friendly format may be displayed to an operator for data entry, but only the entered data is transmitted. These are all functions appropriately handled by the Presentation Layer.

The Process Layer is concerned with higher level functions which provide support to the application or system activities, such as operator support, the use of remote data, file transfer control, or distributed data base activities. When distributed files and data bases are used, for example, controls are needed to prevent integrity problems or deadlocks. Some types of controls for this are strongly related to networking, for example the timestamping of transactions and delivery of transactions in timestamp sequence. Another example of a process layer function would be control of the pace of certain processes so that a transmitting machine can send records continuously without flooding the receiving machine, or so that an application can keep a distant printer going at maximum speed.

3.0 THE SESSION LAYER

The Session Layer of the ISO Standard Layer Model is responsible for managing communications between two 'session-entities'. In an Ada context, a 'session-entity' can be defined as an Ada task. Thus, in an Ada system, the Session Layer manages the rendezvous between two communicating Ada tasks. These tasks may reside on the same or different computing sites. If the communicating tasks reside on different sites, they are said to be distributed. In the AIPS Inter-computer Communications Services (ICCS), the distributed rendezvous function is performed by the Synchronous Communication Manager (SCM). Further, the SCM is responsible for accessing distributed global data objects. The actual location of the distributed tasks or data is transparent to the application.

The SCM also handles any error conditions encountered during task rendezvous or global data access. These are usually reported back to the application as a system or application-defined Ada exception.

In addition to the SCM, the Session Layer may contain application-specific communication protocols for controlling specific task rendezvous (e.g., guard conditions on select statements, etc.)

This section presents the SCM functional requirements and design for two distributed processes: task rendezvous and global data access. The SCM has not yet been implemented on the current AIPS system.

3.1 Synchronous Communication Manager

3.1.1 Functional Requirements

The SCM shall:

1) provide an interface between the Ada run time system and the User Services to enable an Ada application to perform a simple rendezvous with parameter passing between two distributed tasks.

2) provide an interface between the Ada run time system and the User Services to enable an Ada application to perform a conditional rendezvous with parameter passing between two distributed tasks. If the receiving or server task is not available for rendezvous, the sending or client task shall execute the alternate action.

3) provide an interface between the Ada run time system and the User Services to enable an Ada application to perform a timed rendezvous with parameter passing between two distributed tasks. If the receiving or server task is not

available for rendezvous within the specified time interval, the sending or client task shall execute the alternate action.

4) access distributed global data in a contention protected manner.

5) provide Ada exception handling in a distributed environment.

3.1.2 Functional Design

The following sections present the top level design of the Synchronous Communication Manager.

3.1.2.1 Distributed Ada Objects

The design of the SCM supports two kinds of distributed Ada objects: tasks and data. Practically, the actual objects of distribution are packages in which the tasks and data are declared.

The mechanism for distributing these objects is represented in Figure 3-1. This illustration shows the address space for a typical computing site or FTP. The physical address space of the site is a subset of the total logical address space and is the same for all sites. The load module for each FTP is linked in the logical address space and consists of all local objects and the entire set of distributed objects (tasks and data). Each distributed object resides in a unique location within the logical address space. At load time, the physical address space of each site includes: (1) its local objects, (2) the specifications of all the distributed tasks, and (3) only the task bodies and global data packages of those locally resident distributed objects. When an FTP is initialized, its local Memory Management Unit (MMU) is configured to translate the logical address space to the local physical address space for the locally resident distributed objects. Furthermore, this local MMU is set up to interrupt the FTP when any non-resident distributed object is accessed. In this manner, all the distributed objects are visible to each FTP.

3.1.2.2 User Interface

Ideally, the accessing of distributed objects should be transparent to the user. In order for this function to be performed in an efficient manner, it requires compiler support. Since AIPS does not have this support, the user is constrained to access distributed tasks and data in a specific, non-transparent manner.

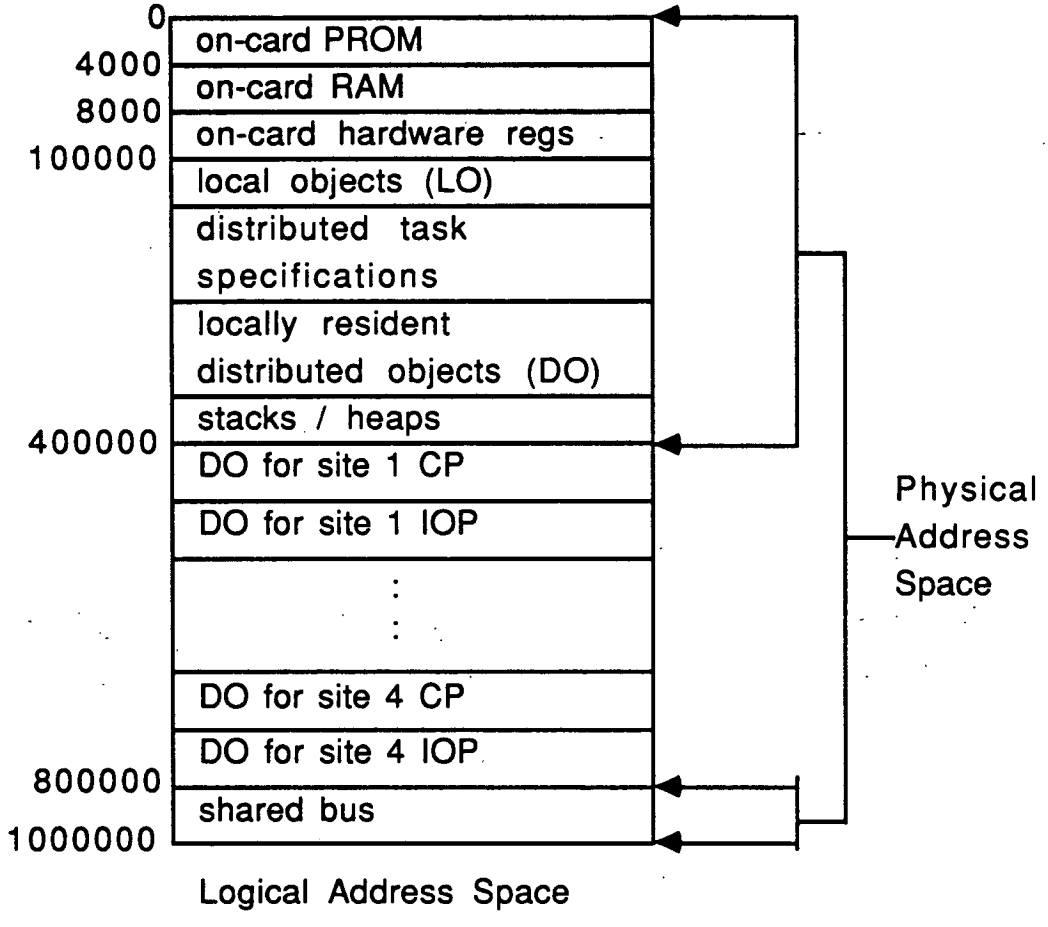


Figure 3-1. FTP Address Space

3.1.2.2.1 Tasks

To permit the specification of a distributed task to the AIPS system, the package specification of a distributed task should be in the following form:

```
package Task_Package is
        .
        .
        -- Task specification
        task type Task_Name_t is
        .
        .
        end Task_Name_t;

        -- task creation
        Task_Name: Task_Name_t;

        -- define ICCS User ID
        My_User_ID: constant ic_user_id := Task_User_ID;

        -- Get  run time system task ID (address of task control block)
        function Get_Task_ID is new LSS_Task_IDs.IDOf(Task_Name_t);
        Task_Name_ID: LSS_Task_IDs.Task_ID:= Get_Task_ID(Task_Name);

        -- Identify distributed task to ICCS
        B: boolean := Identify_Task_Location(My_User_ID,        --user ID
                                             Task_Site,        --task site ID
                                             Task_Processor,   --CP or IOP
                                             Task_Name_ID);    --task ID
        .
        .
end Task_Package;
```

Refer to Sections 4.1.1 and 4.2.1.3.3 for a description of the Identify_Task_Location function. The ICCS User ID identifies the task to the IC Communications Service and is valid throughout all computing sites (see Section 4.2.1.1 and Appendix B for a description of ICCS User IDs). The Task ID identifies the task to the local run time and is unique to each site.

All distributed task entry declarations are of the form:

```
type param_mode is (in, out, inout);

entry entry_name(
                    no_of_param     :    in integer;
                    param_1         :    in system.address;
                    param_1_size    :    in integer;
                    param_1_mode    :    in param__mode;
                          .
                          .
                          .
                    param_n         :    in system.address;
                    param_n_size    :    in integer;
                    param_n_mode    :    in param_mode
                   );
```

where

no_of_param	=	number of formal parameters to be passed
param_n	=	address of nth parameter
param_n_size	=	size of nth parameter in bytes
param_n_mode	=	mode of nth parameter

The parameter no_of_param is required. If no parameters are being passed, it should be set to 0.

3.1.2.2.2 Data

Global data can be accessed using the following procedure:

```
type access_mode is (read, write);

procedure Global_Access(
            mode          in access_mode;
            object:       in system.address;
            object_size:  in integer;
            data_addr:    in system.address;
             );
```

where

mode	=	type of access (read or write)
object	=	address of global object to be accessed
object_size	=	size of object to be accessed ('size attribute)
data_addr	=	address where data is read to or written from

3.1.2.3 Site Initialization

Because each site has the task specifications for all distributed tasks, a task control block (TCB) for each distributed task in the system will be created, at elaboration, on each site. However, only those TCBs on the site on which the corresponding task bodies reside will be active. The remaining TCBs will be inactive and merely contain task location information. The ICCS task location directories (CP_Task_Location, IOP_Task_Location) will also be created at this time via the call to the Identify_Task_Location function.

No directory is needed for global data. The location of any particular variable is determined when it is accessed. If the global data resides on a remote FTP, this determination is made via MMU interrupt (see Section 3.2.2.1).

3.1.2.4 Distributed Task Rendezvous

The top level dataflow of a distributed task rendezvous is presented in Figure 3-2. In such a scenario, a local client task desires to rendezvous with a server task on a remote site. The client task makes the required entry call to the Kernel Entry Procedure (Kernel_Entry). Kernel_Entry checks to see if the desired server task is local or on a remote site. If local, the normal Ada rendezvous is performed. If remote, Kernel_Entry selects and executes a rendezvous with an idle (blocked) server surrogate task from the Surrogate Task Pool. Any formal rendezvous parameters destined for the remote task are passed to the surrogate task.

The surrogate task assembles the IC rendezvous message parameters including any data to be sent to the remote task (Ada **in, inout** parameter mode). This message is transmitted to the Rendezvous Manager Task (Rndv_Mgr) resident on the remote site via the User Services of the Transport Layer. The surrogate task is then blocked waiting for a return message which signals the completion of the rendezvous.

The IC message arrives at the remote site and is passed to the resident Rendezvous Manager Task via a message received event. The Rndv_Mgr selects and performs a rendezvous with an idle client surrogate task from the Surrogate Task Pool. The surrogate task constructs a normal Ada rendezvous calling sequence from the received IC message parameters and performs the desired rendezvous with the server task. After the rendezvous has completed, the surrogate task assembles the IC end-of-rendezvous message parameters including any parameters to be returned from the server task to the client task (Ada **out, inout** parameter mode). This message is then transmitted back to the Rendezvous Manager Task on the client task's site via the User Services. After the end-of-rendezvous message is sent, the client surrogate task is returned to the local Surrogate Task Pool (becomes 'blocked').

Back on the client task's FTP, the IC end-of-rendezvous message is received and the Rendezvous Manager Task is started via the message received event. The Rendezvous Manager Task then performs a rendezvous with the blocked server surrogate task. The surrogate task returns to the client task via the run time kernel and passes any formal rendezvous output parameters received from the server task. The server surrogate task is subsequently returned to the local Surrogate Task Pool.

Assigning unique surrogate tasks to each rendezvous request allows multiple requests to proceed concurrently (subject to priority) without requiring each rendezvous to go to completion before commencing another. Exhaustion of the surrogate task pool on either site results in an exception being raised in the client task.

3.1.2.5 Distributed Data Access

The top level dataflow of a distributed global data access is presented in Figure 3-3. As in the distributed task rendezvous process, the design is based on utilizing surrogate tasks from the Surrogate Task Pool. A local task that wishes to access distributed global data potentially residing on a remote site calls the Global Object Access Procedure (Global_Access). A check is made to determine if the desired data object is local or remote. If local, the access is performed using the resident Local Object Access Procedure (Local_Access) in Local System Services. If remote, a rendezvous is performed with an idle local access surrogate task selected from the Surrogate Task Pool. The access parameters are passed to the surrogate task.

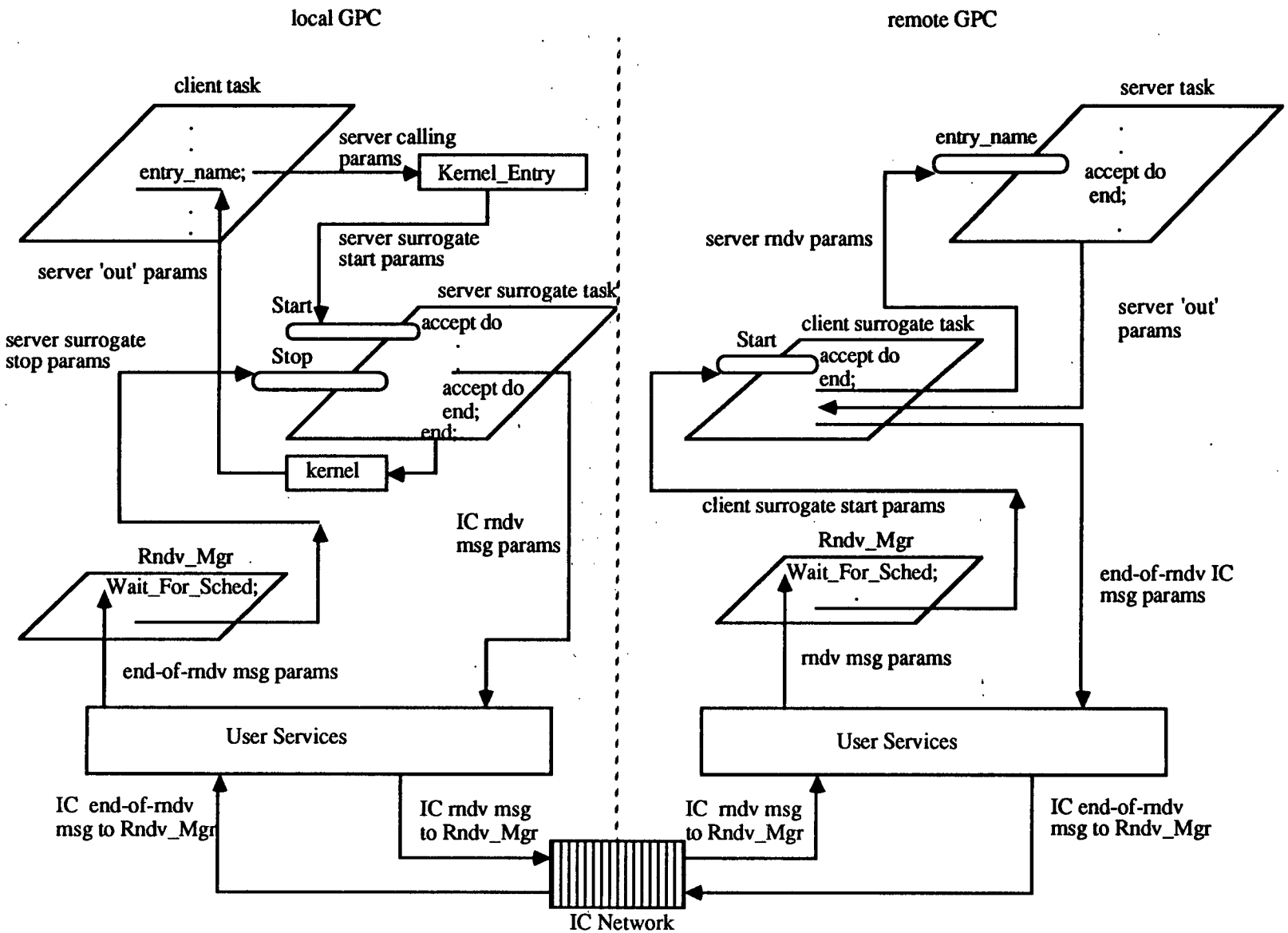


Figure 3-2. Distributed Task Rendezvous

The surrogate task assembles the IC access message parameters which includes any data to be written to the global data object (if write mode). This message is transmitted via the User Services to the Global Data Manager Task (GlobalData_Mgr) that is resident on the remote site. The surrogate task is then blocked waiting for a return message which signals the completion of the access.

The IC message arrives at the remote site and is passed to the resident Global Data Manager Task via a message received event. GlobalData_Mgr selects and performs a rendezvous with an idle remote access surrogate task from the Surrogate Task Pool. The surrogate task accesses the global data item via the resident Local Object Access Procedure (Local_Access). After access has been completed, the surrogate task assembles the IC end-of-access message parameters which includes any parameters to be returned to the local task (if read mode). This message is then transmitted via the User Services to the Global Data Manager Task on the local site. The remote access surrogate task is subsequently returned to the local Surrogate Task Pool (becomes 'blocked').

Back on the local task site, the IC end-of-access message is received and the Global Data Manager Task is started via the message received event. GlobalData_Mgr performs a rendezvous with the blocked local access surrogate task The surrogate task returns to the local task along with any global data object value (read mode). The local access surrogate task is then returned to the local Surrogate Task Pool.

As with the Distributed Task Rendezvous function, the assignment of unique surrogate tasks to each access request allows multiple requests to proceed concurrently (subject to priority) without requiring each access to complete before commencing another. Contention protection is provided by the Local Object Access Procedure resident on the remote site. Exhaustion of the surrogate task pool on either site results in an exception being raised on the local site.

3.2 Synchronous Communication Manager: Software Specifications

The Synchronous Communication Manager consists of the following components:

Distributed Rendezvous
- Kernel Entry Procedure
- Server Surrogate Task
- Client Surrogate Task
- Rendezvous Manager

Distributed Access
- Global Object Access Procedure
- Local Access Surrogate Task
- Remote Access Surrogate Task
- Global Data Manager Task

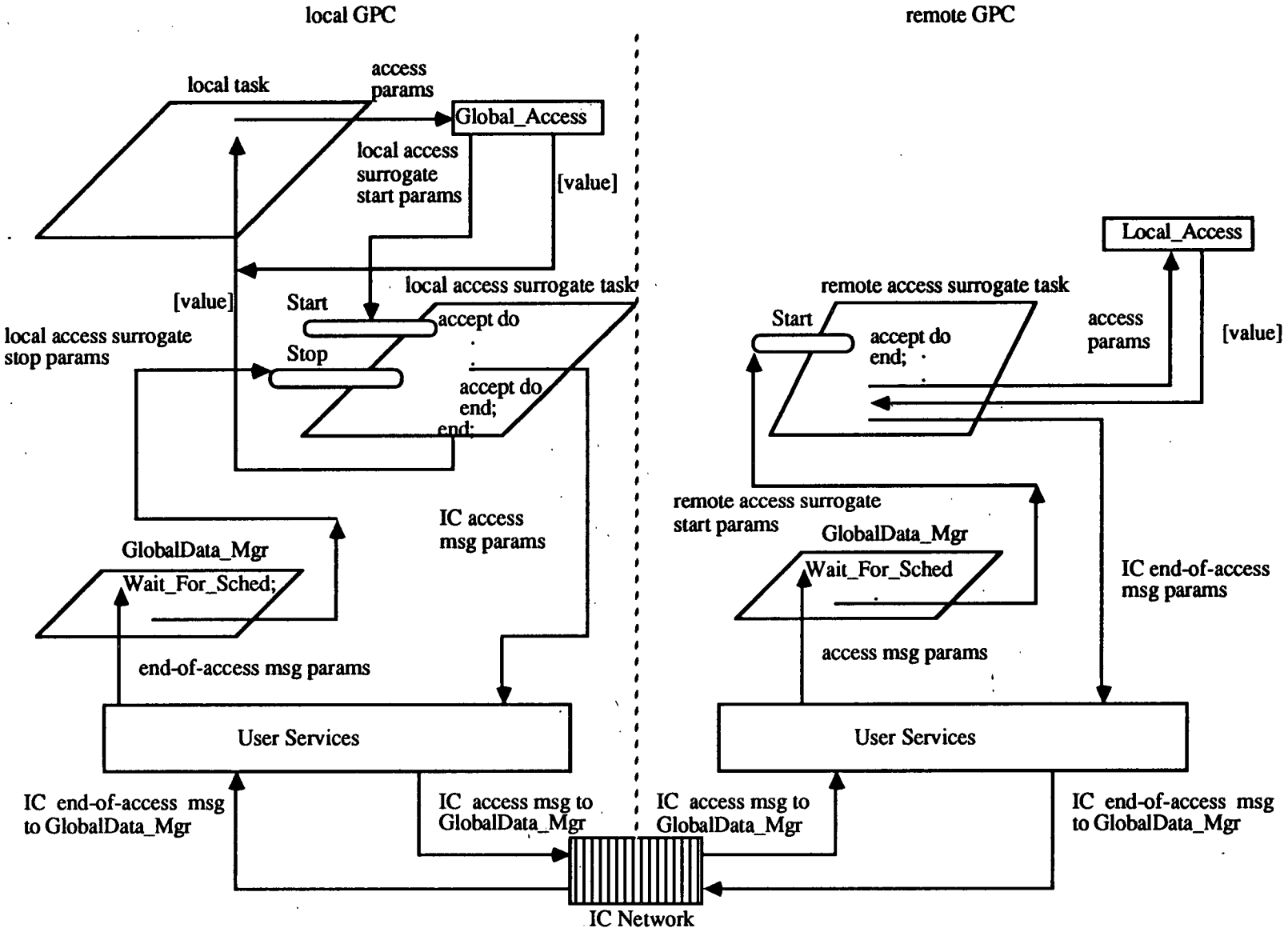


Figure 3-3. Distributed Data Access

3.2.1 Distributed Rendezvous Process Descriptions

3.2.1.1 Process Name: Kernel Entry Procedure

Inputs:

- server calling parameters:
  - server_task_ID
  - server_entry_ID
  - server formal rendezvous parameters
  - entry_type
  - [client_wakeup_time]

Outputs:

To server surrogate task

- server surrogate start parameters:
  - server_task_ID
  - server_entry_ID
  - server formal rendezvous parameters
  - entry_type

or

To server task

- server formal rendezvous parameters

This procedure (Figure 3-4) provides the interface between the application task and the run time kernel for a rendezvous entry call. Three types of Ada rendezvous are supported: simple, conditional, and timed. Upon entry to the kernel, a check (Test_Remote) is made to determine if the desired server task is local or remote. This is determined by accessing the task body start address in the local TCB for the server task. Since all tasks are linked in the logical address space (see Section 3.1.2.1), the task body start address of a remote task is not in the local physical address space and causes an MMU interrupt. If the server task is local, the normal Ada rendezvous kernel routine (ts_call) is executed.

If the server task is remote, Task_Select selects an idle ('blocked') server surrogate task from the Surrogate Task Pool and a local rendezvous is made via ts_call. The original server formal rendezvous parameters are passed to the Start entry of the surrogate task. Thus, the remote rendezvous of client with server has been changed to a local rendezvous of client with server surrogate.

The number of server surrogate tasks in the Surrogate Task Pool determines the maximum number of concurrent remote task rendezvouses permitted. If there are no idle (i.e., blocked at the Start entry) server surrogate tasks in the pool, a TASKING exception is raised in the client task.

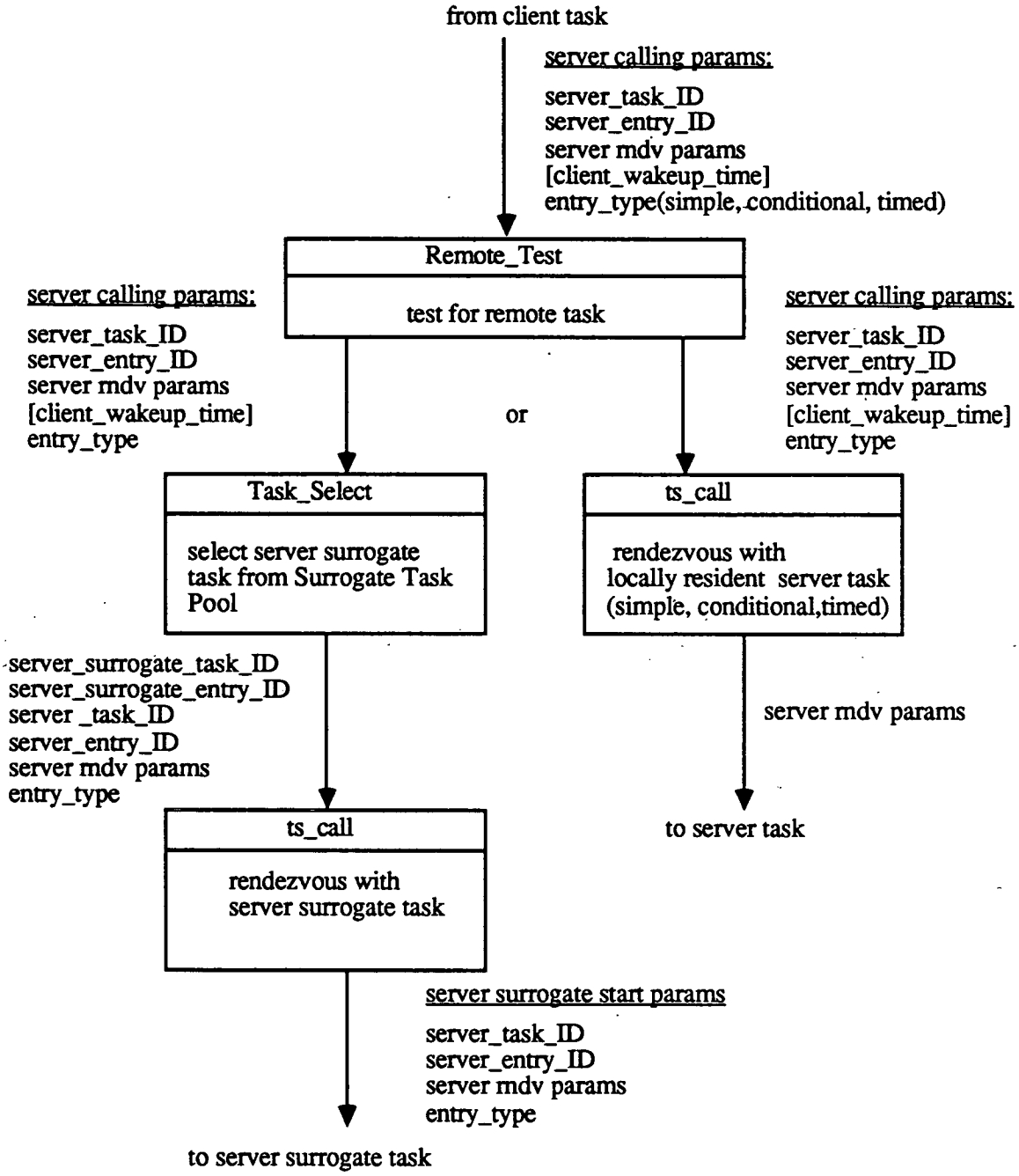


Figure 3-4. Kernel Entry Procedure

3.2.1.2 Process Name: Server Surrogate Task

Inputs:

Start entry

- server surrogate start parameters:
  - server_task_ID
  - server_entry_ID
  - server formal rendezvous parameters
  - entry_type

Stop Entry

- rndv_status
- [server formal rendezvous **out** parameters]

Outputs:

To User Services

- IC rendezvous message parameters

To kernel

- rndv_status
- [server rendezvous **out** parameters]

This task (Figure 3-5) provides the interface between the client task and the User Services in the Transport Layer. The Start entry receives the server task calling parameters from the kernel. The IC rendezvous message is constructed and transmitted to the remote site by the Send_Output procedure of User Services (see Section 4.2.1.3.4) utilizing the following parameters:

- dest_gpc Destination site identifier obtained from the task location directories (CP_Task_Location, IOP_Task_Location).
- dest_task Message destination on remote site (Rendezvous Manager Task).
- msg_size Size of message data buffer.
- msg_prio Priority of message.
- msg_buffer Message data buffer:
  - msg_ID (= entry_type)
  - server_surrogate_task_ID
  - server_user_ID (from task location directories)
  - server_entry_ID
  - server formal rendezvous parameters

Since the **in, inout** server rendezvous parameters are given in terms of local reference addresses, these are converted to the parameter values for transmission to the remote site.

After completion of the call to Send_Output, the surrogate task is blocked at the Stop entry waiting for completion of the rendezvous on the remote site.

The Stop entry receives the rendezvous status indicator (rndv_status) and any server formal rendezvous **out** parameters from the Rendezvous Manager Task (Rndv_Mgr). The rendezvous status indicator tells the kernel whether or not the rendezvous was successfully completed. The **out** parameters are passed to the client task via the kernel during normal completion of the client-surrogate rendezvous. After the rendezvous has completed, the surrogate task is blocked at the Start entry, effectively returning it to the Surrogate Task Pool.

Any exception raised in the server surrogate task is returned to the client task as an Ada TASKING exception.

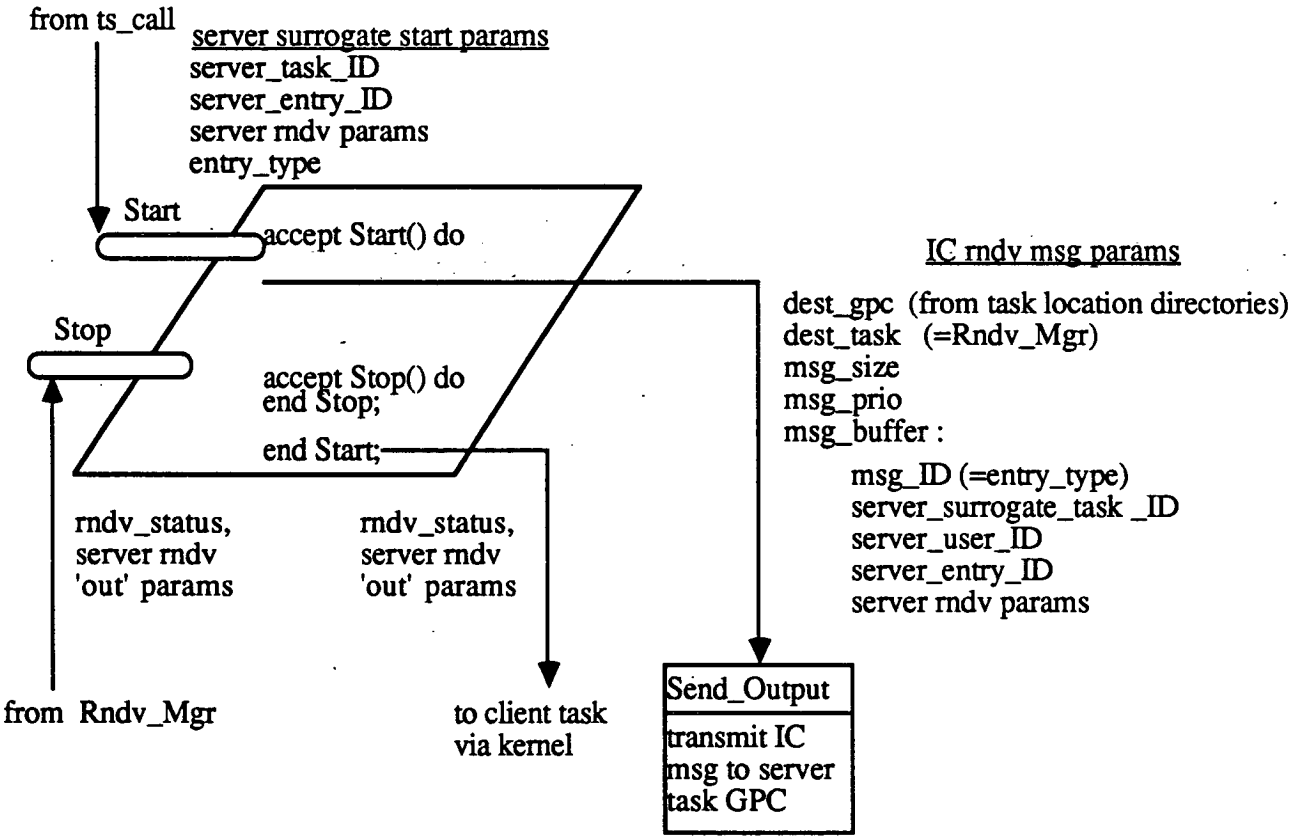


Figure 3-5. Server Surrogate Task

3.2.1.3 Process Name: Client Surrogate Task

Inputs:

Start entry

- client surrogate start parameters:
  - msg_ID
  - source_gpc
  - server_surrogate_task_ID
  - server_user_ID
  - server_entry_ID
  - server formal rendezvous parameters

From server task

- server formal rendezvous **out** parameters

Outputs:

To User Services

- IC end-of-rendezvous message parameters

To server task

- server formal rendezvous parameters

This task (Figure 3-6) provides the interface between User Services in the Transport Layer and the server task in a distributed rendezvous. The Start entry receives the server task calling parameters, rendezvous type (msg_ID) and client task site ID (source_gpc) from the Rendezvous Manager Task. The server formal rendezvous parameters are converted to local address references and a rendezvous is performed with the server task via the local kernel entry procedure (ts_call). After the rendezvous has completed, the IC end-of-rendezvous message is constructed and transmitted to the source site by the Send_Output procedure of User Services utilizing the following parameters:

- dest_gpc Source site identifier (= source_gpc).
- dest_task Message destination on source site (Rendezvous Manager Task).
- msg_size Size of message data buffer.
- msg_prio Priority of message.
- msg_buffer Message data buffer:
  - msg_ID (= end_of_rndv)
  - server_surrogate_task_ID
  - rndv_status
  - server rendezvous **out** parameters

After the completion of the call to Send_Output, the client surrogate task is blocked at the Start entry, effectively returning it to the Surrogate Task Pool.

Any exception raised in the client surrogate task is returned (via rndv_status) to the client task at the source site as an Ada TASKING exception.

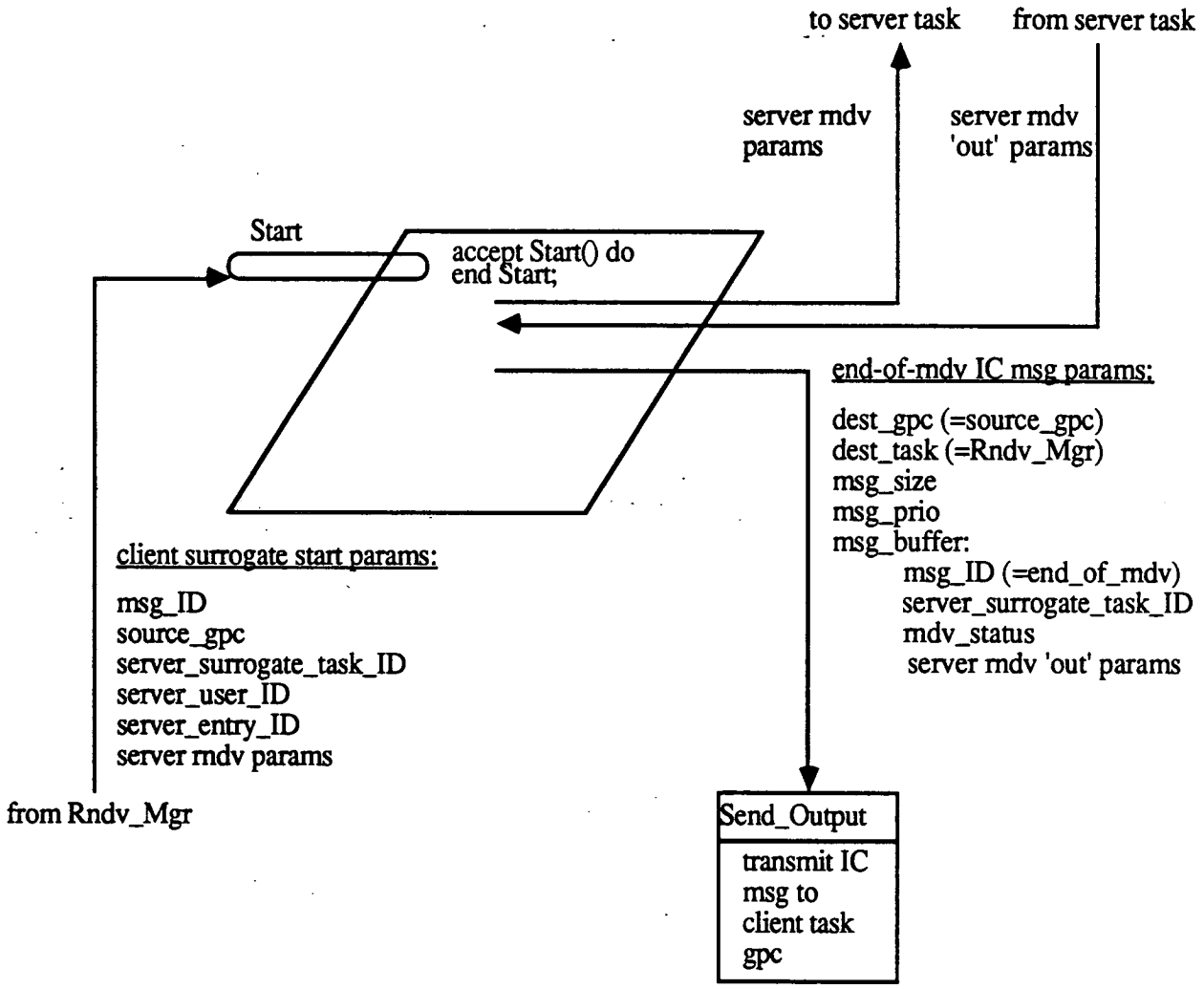


Figure 3-6. Client Surrogate Task

3.2.1.4 Process Name: Rendezvous Manager Task

Inputs:

From User Services

- rendezvous message parameters:
  - msg_ID (= entry_type)
  - server_surrogate_task_ID
  - server_user_ID
  - server_entry_ID
  - server formal rendezvous parameters
  - source_gpc

  or

- end-of-rendezvous message parameters:
  - msg_ID (= end_of_rndv)
  - server_surrogate_task_ID
  - rndv_status
  - server formal rendezvous **out** parameters

Outputs:

To client surrogate task

- client surrogate Start parameters:
  - msg_ID (= entry_type)
  - server_surrogate_task_ID
  - server_user_ID
  - server_entry_ID
  - server formal rendezvous parameters
  - source_gpc

To server surrogate task

- client surrogate Start parameters:
  - rndv_status
  - server formal rendezvous **out** parameters

Upon receipt of an IC rendezvous or end-of-rendezvous message from a remote site, this task (Figure 3-7) is initiated by an event signalled by the Message_Send_Rcv task (see Section 4.2) in the Transport Layer. The message contents are obtained by calling the Get_Input procedure in User Services (see Section 4.2.1.3.5).

If the message received was a rendezvous message, an idle client surrogate task is selected from the Surrogate Task Pool and a rendezvous is made with the Start entry of the

surrogate task. The server task calling parameters, rendezvous type, and source site ID are passed as formal rendezvous parameters.

If the message received was an end-of-rendezvous message, a rendezvous is made with the Stop entry of the server surrogate task (server_surrogate_task_ID) handling the remote rendezvous. The rendezvous status indicator and any server formal rendezvous **out** parameters are passed as formal rendezvous parameters.

The above Start and Stop rendezvous entries are only for synchronization purposes; no processing is performed during the rendezvous and control is returned immediately to the Rendezvous Manager Task. The actual remote rendezvous processing is controlled by the surrogate tasks, leaving the Rendezvous Manager Task free to concurrently handle any other remote rendezvous requests.

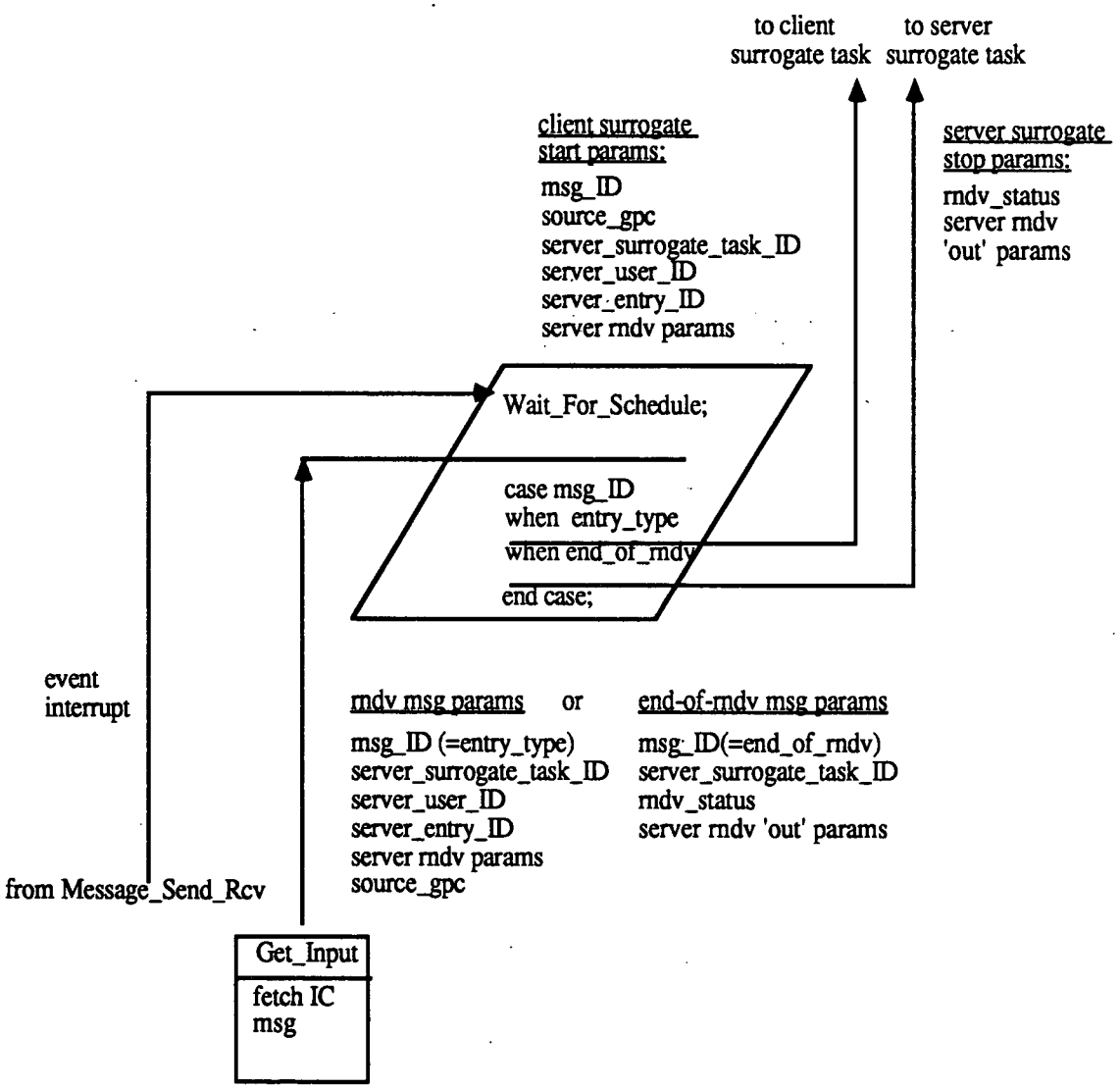


Figure 3-7. Rendezvous Manager Task

3.2.2 Distributed Access Process Descriptions

3.2.2.1 Process Name: Global Object Access Procedure

Inputs:

- access parameters:
  - access_type
  - object_address
  - object_size
  - [object_value]

Outputs:

To local access surrogate task

- local access surrogate Start parameters:
  - access_type
  - object_address
  - object_size
  - [object_value]
  - object_gpc

To local task

- [object_value]

This procedure (Figure 3-8) allows an application to access a distributed global data object without knowing its site location. Upon entry, a check (Remote_Test) is made to determine if the desired data object is local or remote. This is determined by attempting to access the object. Since all global data packages are linked in the logical address space (see Section 3.1.2.1), the address of a global data object that resides on a remote FTP is not in the local physical address space and causes an MMU interrupt. In contrast, if the data object is local, the object is accessed via the Local Access Procedure (Local_Access).

If the data object is remote, an idle local access surrogate task is selected from the Surrogate Task Pool and a rendezvous is made with the Start entry of the surrogate task via the kernel entry call procedure ts_call. The object access parameters and source site ID are passed as formal rendezvous parameters.

The number of local surrogate tasks in the Surrogate Task Pool determines the maximum number of concurrent global data accesses permitted. If there are no idle (i.e., blocked at Start entry) local access surrogate tasks in the pool, a TASKING exception is raised in the local task.

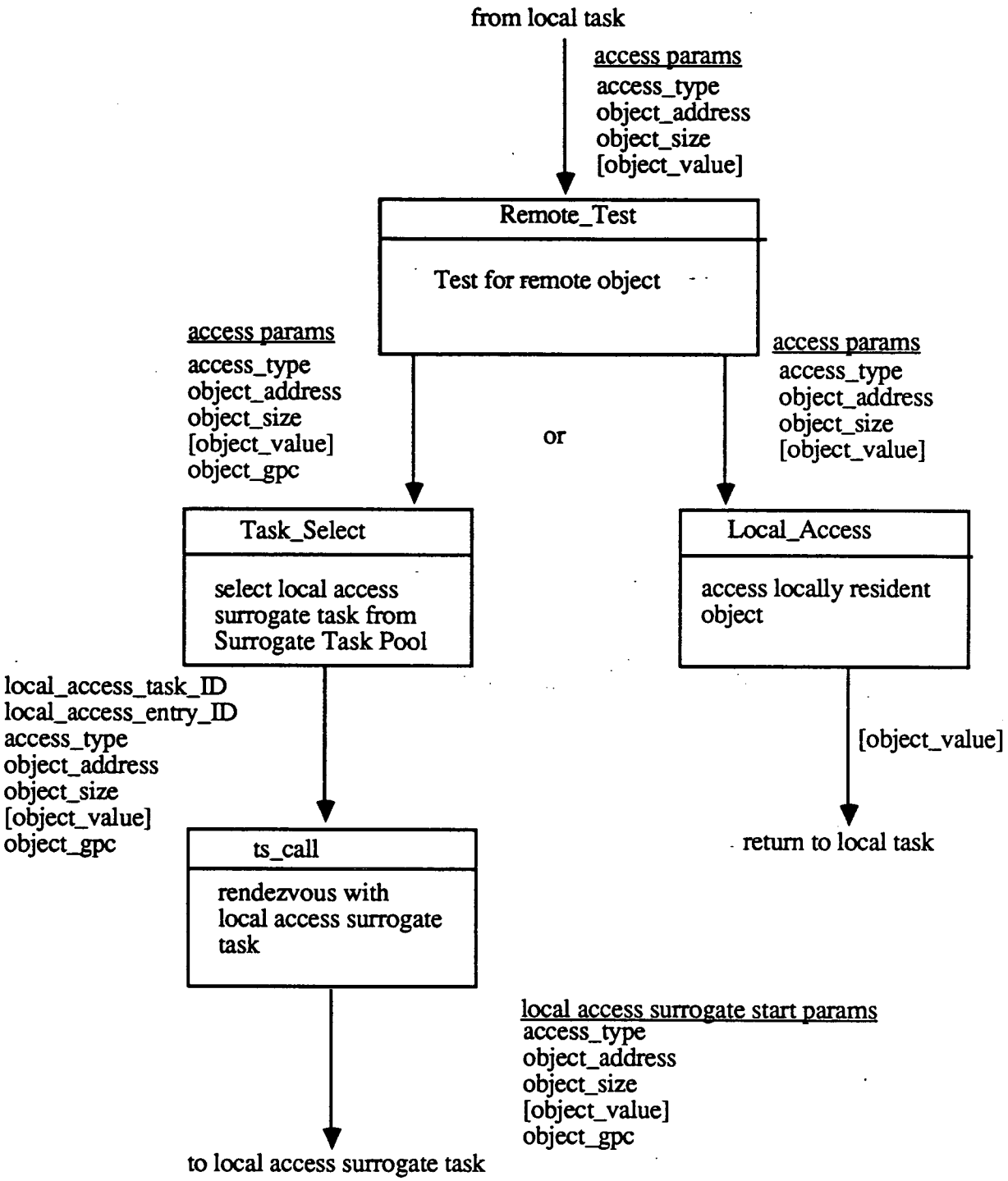


Figure 3-8. Global Object Access Procedure

3.2.2.2 Process Name: Local_Access Surrogate Task

Inputs:

Start entry

- local access surrogate start parameters:
  - access_type
  - object_address
  - object_size
  - [object_value]
  - object_gpc

Stop Entry

- local access surrogate stop parameters:
  - [object_value]

Outputs:

To User Services

- IC access message parameters

To kernel

- [object_value]

This task (Figure 3-9) provides the interface between the local task and the User Services in the Transport Layer. The Start entry receives the object access parameters from the kernel. The IC access message is constructed and transmitted to the remote site by the Send_Output procedure of User Services (see Section 4.2.1.3.4) utilizing the following parameters:

- dest_gpc Destination site identifier (= object_gpc)
- dest_task Message destination on remote site (Global Data Manager Task).
- msg_size Size of message data buffer.
- msg_prio Priority of message.
- msg_buffer Message data buffer:
  - msg_ID (= access_type)
  - local_access_task_ID
  - object_address
  - object_size
  - [object_value]

Since global object addresses are valid on the remote site, these may be transmitted directly. After completion of the call to Send_Output, the surrogate task is blocked at the Stop entry to wait for completion of the data access at the remote site.

The Stop entry receives any global object value (read mode) from the Rendezvous Manager Task (Rndv_Mgr). The object value is passed to the local task via the kernel during normal completion of the local task-surrogate rendezvous. After the rendezvous has completed, the surrogate task is blocked at the Start entry, effectively returning it to the Surrogate Task Pool.

Any exception raised in the local access surrogate task is returned to the local task as an Ada TASKING exception.

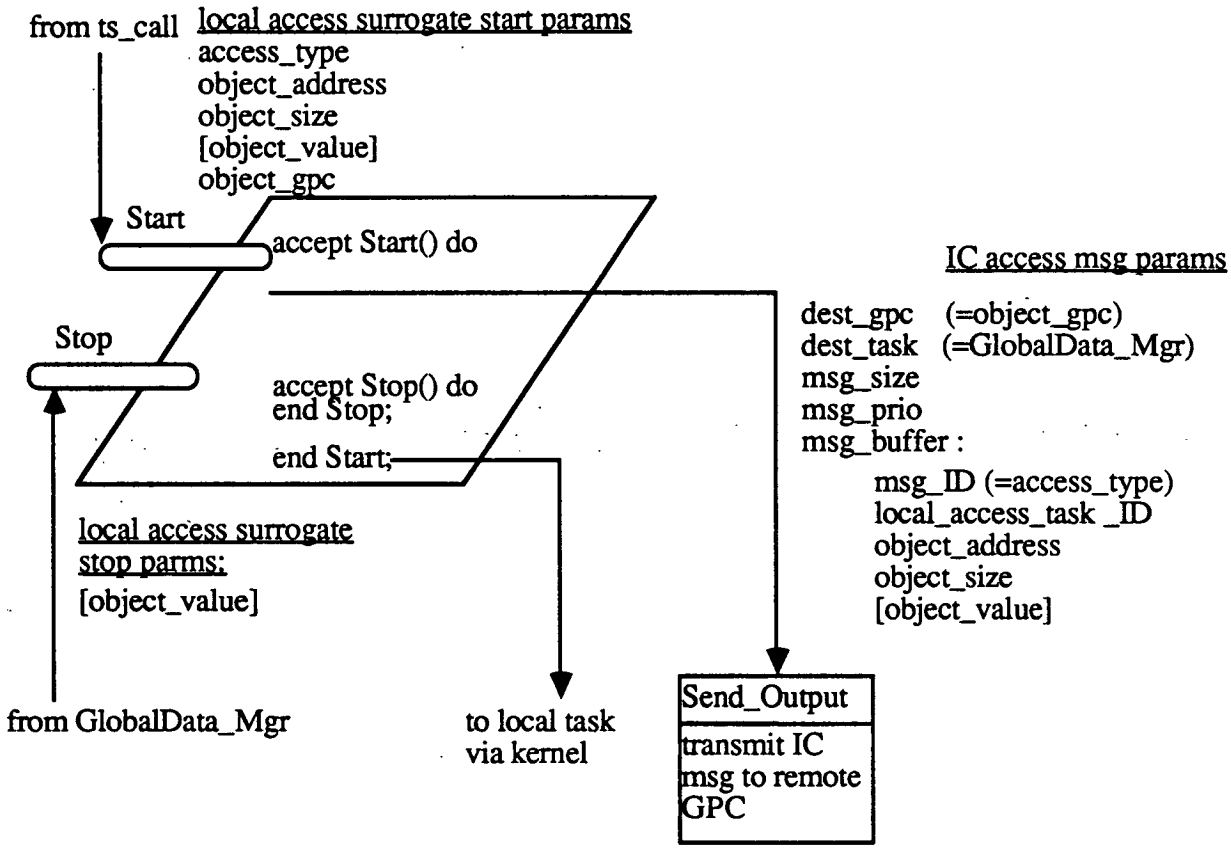


Figure 3-9. Local Access Surrogate Task

3.2.2.3 Process Name: Remote Access Surrogate Task

Inputs:

Start entry

- remote access surrogate start parameters:
  - msg_ID
  - source_gpc
  - local_access_task_ID
  - object_address
  - object_size
  - [object_value]

From Local Access

- [object_value]

Outputs:

To User Services

- IC end-of-access message parameters

To Local Access

- access parameters:
  - access_type
  - object_address
  - object_size
  - [object_value]

This task (Figure 3-10) provides the interface between User Services in the Transport Layer and the Local Access Procedure (Local_Access) in a distributed global data access. The Start entry receives the local access parameters, access type (msg_ID), and source site ID (source_gpc) from the Global Data Manager Task. The locally resident global data is accessed via a procedure call to Local_Access. The data is accessed in a contention protected manner. After the access has completed, the IC end-of-access message is constructed and transmitted to the source site by the Send_Output procedure of User Services utilizing the following parameters:

- dest_gpc Source site identifier (= source_gpc).
- dest_task Message destination on source site (Global Data Manager Task).
- msg_size Size of message data buffer.
- msg_prio Priority of message.
- msg_buffer Message data buffer:
  - msg_ID (= end_of_access)
  - local_access_task_ID
  - [object_value]

After completion of the call to Send_Output, the remote access surrogate task is blocked at the Start entry, effectively returning it to the Surrogate Task Pool.

Any exception raised in the remote access surrogate task is returned to the access requesting task at the source site as an Ada TASKING exception.

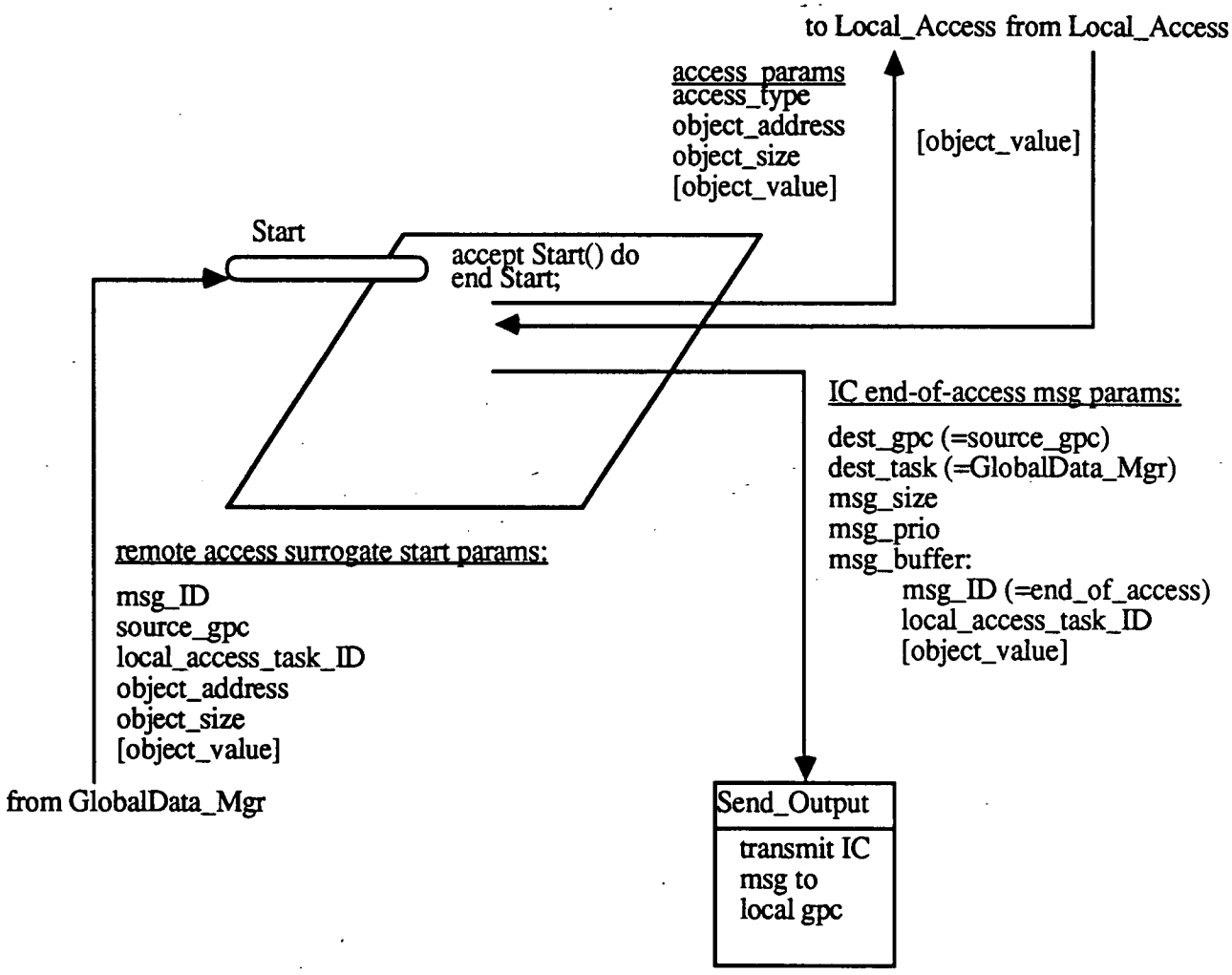


Figure 3-10. Remote Access Surrogate Task

3.2.2.4 Process Name: Global Data Manager Task

Inputs:

From User Services

- access message parameters:
  - msg_ID (= access_type)
  - local_access_task_ID
  - object_address
  - object_size
  - [object_value]
  - source_gpc

  or

- end-of-access message parameters:
  - msg_ID (= end_of_access)
  - local_access_task_ID
  - [object_value]

Outputs:

To remote access surrogate task

- remote access surrogate Start parameters:
  - msg_ID (= access_type)
  - local_access_task_ID
  - object_address
  - object_size
  - [object_value]

To local access surrogate task

- local access surrogate Stop parameters:
  - [object_value]

Upon receipt of an IC access or end-of-access message from a remote site, this task (Figure 3-11) is initiated by an event signalled by the Message_Send_Rcv task (see Sections 4.2.1 and 4.2.2.2.28) in the Transport Layer. The message contents are obtained by calling the Get_Input procedure in the Transport Layer.

If the message received was a access message, an idle remote access surrogate task is selected from the Surrogate Task Pool and a rendezvous is made with the Start entry of the surrogate task. The local access parameters and source site ID are passed as formal rendezvous parameters.

If the message received was an end-of-access message, a rendezvous is made with the Stop entry of the local access surrogate task (local_access_task_ID) handling the remote access. Any access output value (read type) is passed as a formal rendezvous parameter.

The above Start and Stop rendezvous entries are only for synchronization purposes; no processing is performed during the rendezvous and control is returned immediately to the Global Data Manager Task. The actual remote access processing is controlled by the surrogate tasks, leaving the Global Data Manager Task free to concurrently handle any other remote access requests.

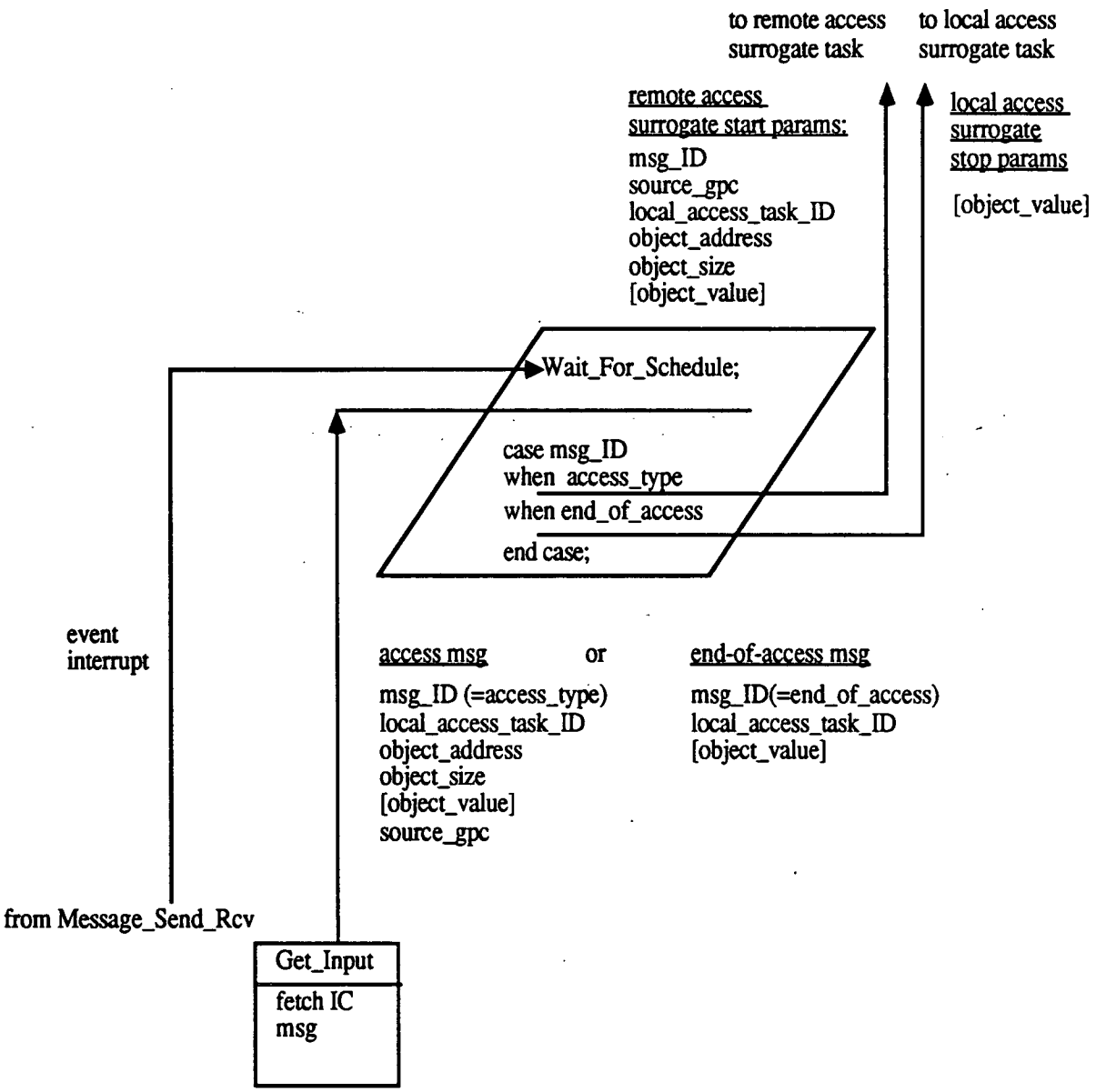


Figure 3-11. Global Data Manager Task

4.0 THE TRANSPORT LAYER

4.1 Functional Requirements and Design

The purpose of the Transport Layer is to provide end-to-end data transfer which is reliable, efficient and transparent to a user. The functions provided by this layer include producing a correct message in the presence of physical transmission errors, controlling the flow of messages so that they are not overwritten before being processed, and servicing multiple users who may be attempting to transmit messages simultaneously.

As shown in Figure 4-1, the Transport Layer as designed for AIPS has several sub-layers. At the highest level are the User Services, whose function is to provide a standard interface to the Transport Layer user which is simple and insulates the user from the details of how the lower sub-layers are implemented. Below the User Services is the Message Send Receive function, which transmits outgoing messages that have been funnelled to it by the User Services and distributes incoming messages to the appropriate user. Below the Message Send Receive process on the input side is the ICIS Redundancy Management function, whose job is to analyze the redundant copies of an input message available in the ICIS, identify faults, and present a single congruent copy of the message to the Message Send Receive function.

The Transport Layer must be able to service users residing on both processors of an FTP channel, as well as multiple users on the same processor. To serve these diverse users most efficiently, the User Services are designed as a set of re-entrant subroutines which may be called by any Transport Layer user, while the Message Send-Receive process is a separate task. Communication between the two sub-layers takes place through a set of shared data structures that allow output messages to be queued for the Message Send-Receive task and input messages to be queued for the user. The format of the data structures and the program design allow the two sub-layers to operate asynchronously. Similarly, the ICIS Redundancy Management function is designed as a separate task which passes congruent input messages to the Message Send-Receive task through a shared data structure. The format of this data structure allows these two sub-layers to also operate asynchronously.

4.1.1 Functional Requirements and Design: User Services

A Transport Layer user is any process that wants to send a message without expecting any session layer functions to be performed for it. System tasks such as the Status Reporter and System Manager and applications such as the Central Command and Processing task are examples of users that interface directly with the Transport Layer. The primary services available to these users are:

- sending a message
- confirming that an outgoing message was successfully transmitted, and
- retrieving incoming messages which have previously arrived.

OVERVIEW OF THE TRANSPORT LAYER

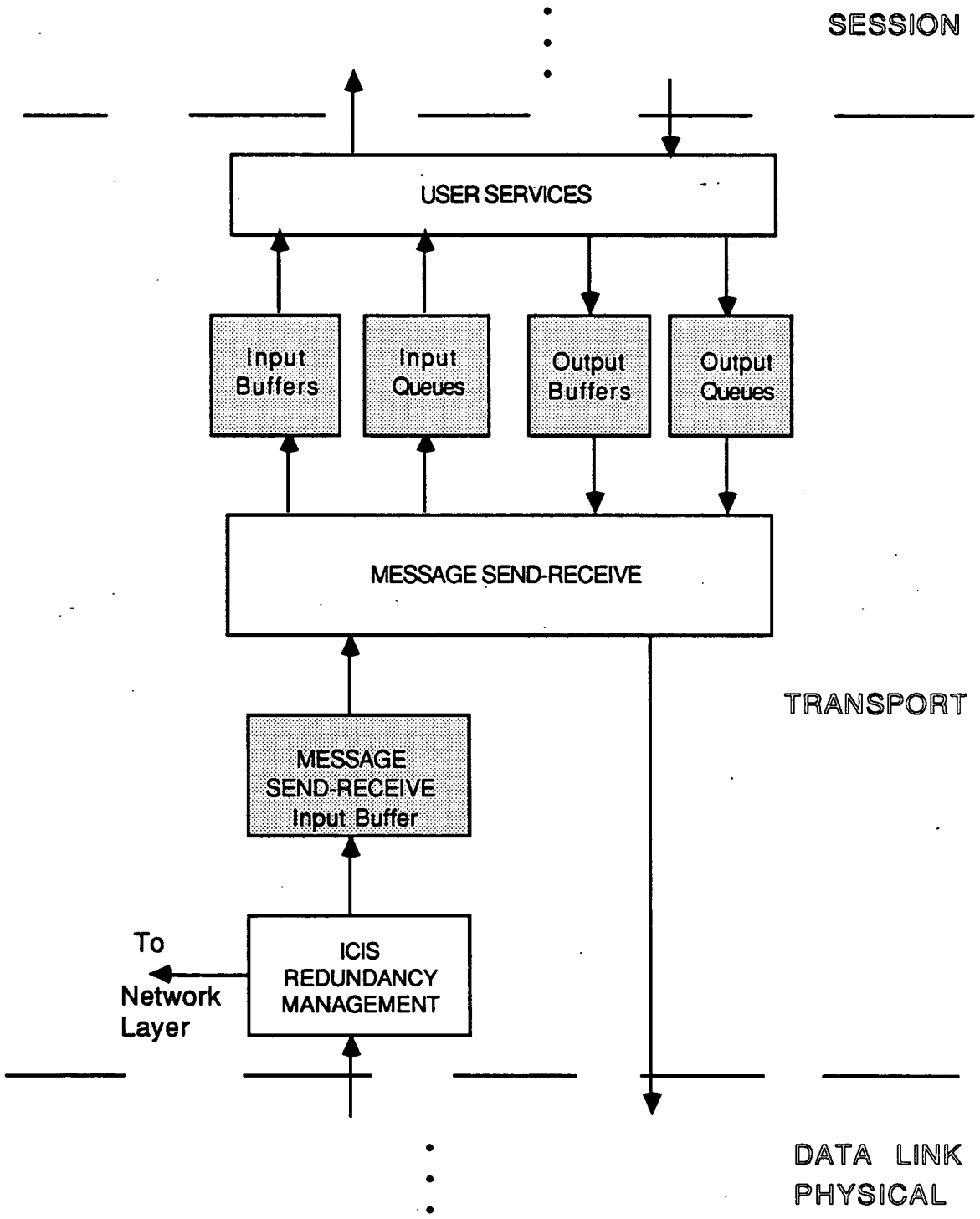


Figure 4-1. Transport Layer Overview

In addition, there are initialization services which a user must invoke before beginning its main task loop. These services enable the user to:

- specify the location (GPC and processor) where it will execute,
- specify the size of an "IN" box, where incoming messages will be held until the user retrieves them, and
- specify the size of an "OUT" box, where outgoing messages will be held until they can be transmitted.

Message Sending. This service allows the user to send a particular message. In addition to identifying the message to be sent, the user must also specify information required to send it, such as the destination GPC and destination user. Point-to-point transmission and broadcasting are both supported. A user may also specify that later on it will want confirmation that the message was successfully transmitted. The user's message and relevant control information required to send it are copied into the user's "OUT" box and the message is added to the outgoing message queue. The Message Send-Receive task is then notified that there is work to do.

Message Retrieval. As they arrive, incoming messages are stored in the destination user's "IN" box. The message retrieval function hands over these accumulated input messages in FIFO order to the user. In addition to the actual message, the user is also provided with the GPC and task identification of the sender and a user-supplied message priority.

Status Checking. This service tells a sender the current status of a particular output message. This feature might be appropriate for a user who is sending a one-time message and wishes to retransmit if the message does not get through on the first attempt. The status information indicates whether the transmission is still in progress or has completed, and if completed, whether successfully or with a particular type of error.

Task Location Identification. This is a one-time service which a user must invoke in order to identify itself to the Transport Layer. Here the user specifies its user ID, processor ID (either CP or IOP), and GPC ID (either a specific GPC or ALL_GPCS). The information is entered into a central database so that a message recipient can be located simply by its user ID, if desired. This database will be updated when functions are migrated from one GPC to another. This shields a user from having to know the specific location of a particular task.

Output Setup. This is a one-time service which must be invoked by users wishing to send output messages. Here the user must specify the size of the largest message it will be sending and the maximum number of output messages it expects to have ongoing at any one time. Buffers of the size and number indicated are dynamically allocated and their addresses stored in an Output Buffer List for the particular user.

Input Setup. This is a one-time service which must be invoked by users expecting to receive input messages. Here the user must specify the size of the largest message it expects to receive and the maximum number of input messages it anticipates may be queued for it an any one time. Buffers of the size and number indicated are dynamically allocated and their addresses stored in an Input Buffer List for the particular user. The user process also specifies whether it wants to be signaled by an event when input arrives or whether it is going to poll for input. If the user wants to be signaled, an event is allocated and initialized and its address returned so that the user may schedule itself based on that event.

4.1.2 Functional Requirements and Design: Message Send-Receive Task

The Message Send-Receive task has two main functions: (1) to send output messages as requested by Transport Layer users and (2) to distribute incoming messages.

The Send function has two main parts. The first part is to physically transmit a message via the ICIS. This involves making a user message conform to the packet size required by the ICIS and executing the ICIS chain required to start the physical transmission. The second part is to guarantee the reliability of a particular communication by using a protocol which confirms the arrival of the message at the end user level.

The Receive function does not read incoming packets from the ICIS; this work is done by the ICIS Redundancy Management task. The Receive function takes the congruent packets which have been formed by ICIS Redundancy Management, assembles them into a complete message, and saves the message for the user's later reference. Optionally, it causes the user to be signalled when input arrives.

Transforming Messages into Packets. The basic unit of data that can be transmitted and received by the ICIS is a packet, which can be a maximum of 127 bytes. Not only must a user message be divided into sections if it is bigger than the maximum packet size, but control information must be attached to each message (or section of a message) so that the message can be identified at its destination. Additional control information for the ICIS hardware and for the ICIS Redundancy Management task must be attached before a message is ready to be physically transmitted.

Message Sending Protocol. Packets are sent using a limited send-acknowledge protocol. The "send-acknowledge" part of this protocol enables the sending GPC to receive explicit confirmation that each packet was received. This confirms that the destination GPC is functioning correctly; in the case of a simplex GPC it also confirms that the ICIS and communication links are functioning correctly. The "limited" part of the protocol controls the flow of messages so that input packets are not read into the destination ICIS dual-ported memory faster than they can be handled by the processor. Specifically, the number of outgoing packets to any other one GPC at any given time is limited. When the maximum number has been reached, a new output packet cannot be sent until an acknowledgement is

received for a previous one. Messages that do not fit into one packet require a series of transmissions: each packet that is sent must be acknowledged before the next section of the message can be sent.

Output messages are sent in FIFO order. There is no provision for ordering them according to the priority of the originating task or user-assigned priority of the individual message.

Assembling and Distributing Input Messages. Input packets must be stripped of the header information inserted by the sending Transport Layer before the packet is presented to the end user. In the case of messages consisting of multiple packets, the entire message must be assembled, with successive packets being checked for correct sequencing. It is also necessary to check that a user's "IN" box, where the message is held until the user requests it, is not full. In this case, which indicates that the user is far behind in his processing, an Undeliverable indication is returned to the sending GPC.

4.1.3 Functional Requirements and Design: ICIS Redundancy Management (RM)

The hardware modular redundancy scheme used to provide highly reliable processing in the core AIPS FTP is extended to the inter-computer communication hardware for the distributed AIPS system. Three redundant communication paths or layers are available for passing redundant copies of communicated data between FTPs. Each channel of an AIPS FTP has a hardware module, the Inter-Computer Interface Sequencer (ICIS), which provides an interface into the AIPS communication network. Each ICIS provides a means for the channel to transmit on only one of the three redundant communication layers and to receive redundant copies of data packets from all three layers. The ICISes for redundant channels of an FTP transmit on distinct layers; when a triplex transmits on the network under non-faulty conditions, a packet is sourced onto each of the three layers from one and only one of the channels. At the receiving end of this communication path is another set of ICISes, one per channel of the receiving FTP, each of which receives a copy of the communication packet from each active layer. Since the specification for the AIPS distributed system calls for the support of mixed redundancy levels of the FTPs subscribing to communication services, the number of active layers and the number of redundant copies of data packets received are time-varying parameters even in a fault-free system.

Unique functional requirements are made of the system software to support this redundant communication system. That is, there are software functions to be performed in this system which are not necessary to support a non-redundant communication system. Software operations are required to distill a single, channel-congruent representation of a data packet from the multiple redundant packets received. Error detection and fault isolation operations must be performed. Status updates must be made in response to error

detections to assure synchronous, valid operation of the communication hardware. The congruent state of the ICIS hardware must be maintained during the recovery of channels, communication layers, and ICISes following repair operations or after transient faults.

4.1.3.1 Functional Requirements Associated With IC Communications

A software interface is required between the higher level software functions of the ICCS Transport Layer and the redundant ICIS hardware. This software interface hides the redundancy of the communication hardware from the other Transport Layer software. That is, the higher levels of the Transport Layer need no knowledge of the redundancy of incoming packets and are not required to perform any special operations related to the redundant hardware during the transmission of packets.

4.1.3.1.1 Redundancy Management During Data Reception Process

The source congruency function has the responsibility of processing the redundant communication packets received from the IC network such that all redundant channels of the receiving FTP obtain a congruent representation of this data. This function requires that the received data be exchanged among the channels of the receiving FTP and that voting operations be performed for fault masking purposes. Congruent data and congruent decisions about the fault status of this data are required to be returned by this function under any fault scenario which is to be tolerated by the core FTP.

The ICIS RM software is responsible for making all decisions related to the reception of data from the IC network. A channel congruent decision must be made as to whether new data has been received by the ICIS and is ready for processing. A decision must be made as to which layer or layers had data on them during a packet communication event. A congruent decision must be made as to the number of data bytes received within an input packet. All of these decisions are made in support of the source congruency function for the received data.

4.1.3.1.2 Redundancy Management During Data Transmission Process

Software support is required for correct operation of the ICIS hardware during the transmission of data packets on the IC network. Note that this operational phase is assumed to include the network polling function as well as the sourcing of raw data on the network once a subscriber has gained network possession. The correct redundancy level encoding must be updated before an ICIS poll operation is initiated to insure correct operation during the "redundancy contention sequence" of the network poll. The masks for the ICIS state exchange voters must be updated on the basis of the current channel/ICIS configuration. A determination as to which layers will have data transmitted on them must be made in order to correctly encode the layer redundancy level in the outgoing packet (for layer fault detection and isolation purposes), and in order to enable the outputs to the

appropriate layers. A channel congruent decision must be made as to when the ICIS has completed previous transmission activity and is available for transmitting another packet. Each channel must synchronously command its ICIS to begin the polling and data transmission operation.

4.1.3.2 Requirements for the Detection and Isolation of Faults (FDI) Associated With the IC Hardware

The detection of error states in the IC hardware and the isolation of faults related to such states are required functions of the ICIS redundancy management software. These required functions facilitate a more robust communication process. Valid communicated data is more likely to be transferred if hardware faults are detected, isolated, and masked.

4.1.3.2.1 FDI During Data Reception Process

Faults in the IC hardware may result in erroneous data and status values being received at a network site. Comparisons of received redundant copies of data and status values are used to detect the presence of such errors. The location of the fault responsible for the error is isolated to a "fault isolation region" on the basis of these same comparisons of redundant data and status. It is recognized that there are cases in which the redundancy of the received information may not be sufficient for making unambiguous fault isolation decisions. No received data is passed along to the Transport Layer when the ambiguity can not be resolved. But the fact that there has been a disruption in the operation of the IC network will be reported to the System FDIR Manager, which may have access to information required to accurately isolate the fault.

The FDI process is able to determine error patterns indicative of a situation where two or more simultaneous faults have manifested errors. The processing of received data associated with a communication packet exhibiting such "double fault" error patterns can be aborted with no data being passed on to the Transport Layer.

4.1.3.2.2 FDI During Data Transmission Process

The transmission of data from an ICIS onto the IC network requires a successful network polling operation in which the site is granted network possession, followed by the execution of a chain of ICIS instructions responsible for outputting the data. All requests by the Message Send-Receive function and the IC Network Manager for transmitting data must be funnelled through an ICIS RM function. This function is required to detect errors associated with network arbitration and network possession. Errors related to the inability to execute ICIS instruction chains are also detected by this function.

4.1.3.3 Requirements for the Management of Responses to Detected Errors in IC Hardware

The ICIS RM software makes responses to detected errors as required to maximize the likelihood of continued correct communication of data across the IC network. In response to certain detected faults, the ICIS RM notifies other ICCS functions (e.g., Network Manager) of a fault and lets these functions manage subsequent error correction or fault masking operations. The ICIS RM is responsible for corrective actions or fault masking operations involving the local ICIS resource. Status variables reflecting the health and availability of ICISes and layers are maintained and used in masking of data/status during the processing of packets received from the network. Certain ICIS control registers (e.g., "voter" mask for the ICIS state exchange) are also updated in response to detected errors.

In response to detected errors, the ICIS RM executes retry self-tests and evaluates the results of these tests to determine if a fault has been removed or if a transient error condition has subsided. These self-tests are only required to evaluate the state of faulty hardware local to the FTP (i.e., the ICIS itself or inter-ICIS links), and not faults on the actual network (i.e., nodes and inter-nodal links) which are handled by the IC Network Manager.

4.1.3.4 Requirements for Performing Re-Initialization of ICIS Hardware on Recovery of Channel and/or ICIS

The ICIS RM software is responsible for all functions required to bring the ICIS hardware back into operation following the recovery of a previously failed FTP channel and/or ICIS. Some ICIS registers associated with network polling and the execution of solicited chains used for outputting data onto the network are also re-initialized, while other registers are dynamically updated when each request to execute a solicited chain is made. All necessary ICIS registers and ICIS instruction sequences required to receive unsolicited input are initialized following a recovery event. The re-initialization function is implemented such that no unsolicited input to the FTP from the IC network is lost during the course of bringing an ICIS back on-line. This ICIS re-initialization function is also required to manage information regarding the state of the unsolicited input data buffers in ICIS dual-port memory at the time of the re-initialization. This information is needed by the ICIS RM software processing incoming data; a determination must be made as to which received data packets came before and which came after the recovered ICIS was brought on-line. An interface to the Local System Services software is provided so that requests to re-initialize the ICIS of a recovered channel can be made to the ICIS RM software.

The current implementation of the ICIS hardware requires software intervention to recover from a "possession default" event. A possession default is said to have occurred if an ICIS is in a state of network possession and it recognizes network activity in the form of "poll pulses" used to initiate network arbitration sequencing. The ICIS RM software must

provide a function to reset the ICIS state machine following a possession default; otherwise the ICIS will never be able to start another required poll sequence preceding the transmission of data on the network.

4.2 Software Specifications

As mentioned previously, the Transport Layer must be able to support users residing on both processors of an FTP channel. The lower sub-layers of the Transport Layer, i.e., the Message Send Receive and ICIS Redundancy Management functions, execute on only one processor (IOP), while the User Interface routines reside on both the CP and IOP. This allocation of tasks and the location of their shared data is illustrated in Figure 4-2.

The Ada packages used to implement the Transport Layer must be structured so that the CP link does not include modules used only on the IOP. In particular, the User Services must not reference packages used on the IOP, such as the Message Send-Receive package. The Ada packages which make up the Transport Layer are shown in Figure 4-3. The arrows in this figure indicate one package referencing ("with"ing in Ada terminology) another. The end of the arrow represents the package making the reference; the arrow point touches the package being referenced. The long dashed line separates Transport Layer users from the Transport Layer. The short dashed line separates packages located on both the CP and IOP from those located only on the IOP.

4.2.1 Software Specifications: User Services

4.2.1.1 ICCS User IDs

Transport Layer users need a way of identifying themselves, both to each other and to the Transport Layer. The most straightforward way to do this is to define an enumeration type in a package which can then be referenced by all user packages as well as packages in the Transport Layer.

This scheme becomes cumbersome, however, when new users must be added to the system. Each time a new user is added numerous packages must be recompiled, including many system files. In addition, it is desirable to separate system users from application users, so that when new application user IDs are added, only application tasks need to be recompiled. To deal with these issues, a scheme has been adopted which defines some maximum number of user IDs and then separates them into two packages, system user IDs and application user IDs. Each package defines its own IDs in a fixed position and leaves space for the users it cannot see. Intermediate USER_SERVICES packages are then employed to take the user ID as defined for the user (i.e., the enumeration type) and use the 'pos Ada attribute to translate it into the user ID as defined for the Transport Layer (i.e., an integer). This scheme is illustrated in Figure 4-3. The reader should refer to Appendix B: Transport Layer User's Guide for specific details and examples.

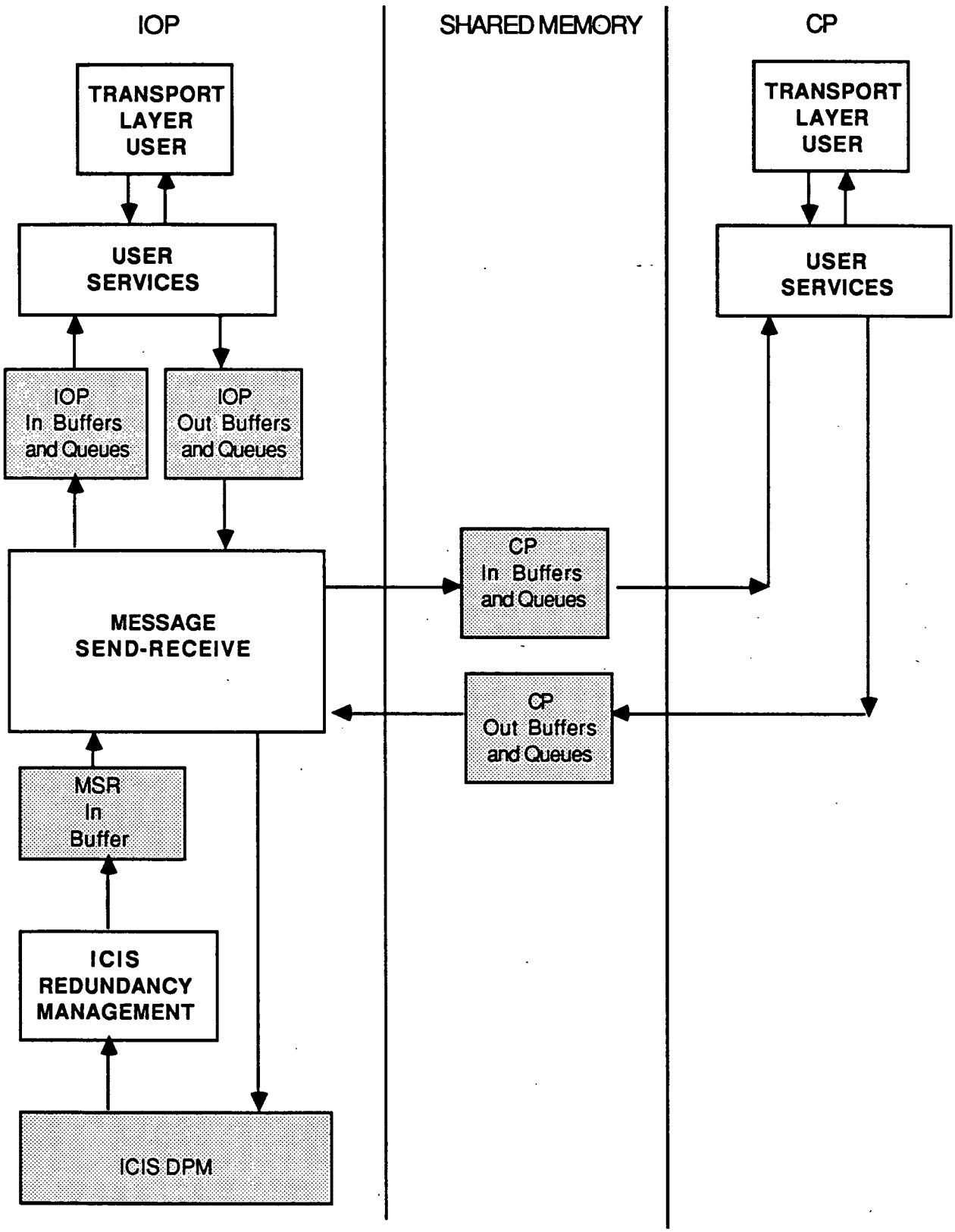


Figure 4-2. Location of Tasks and Shared Data

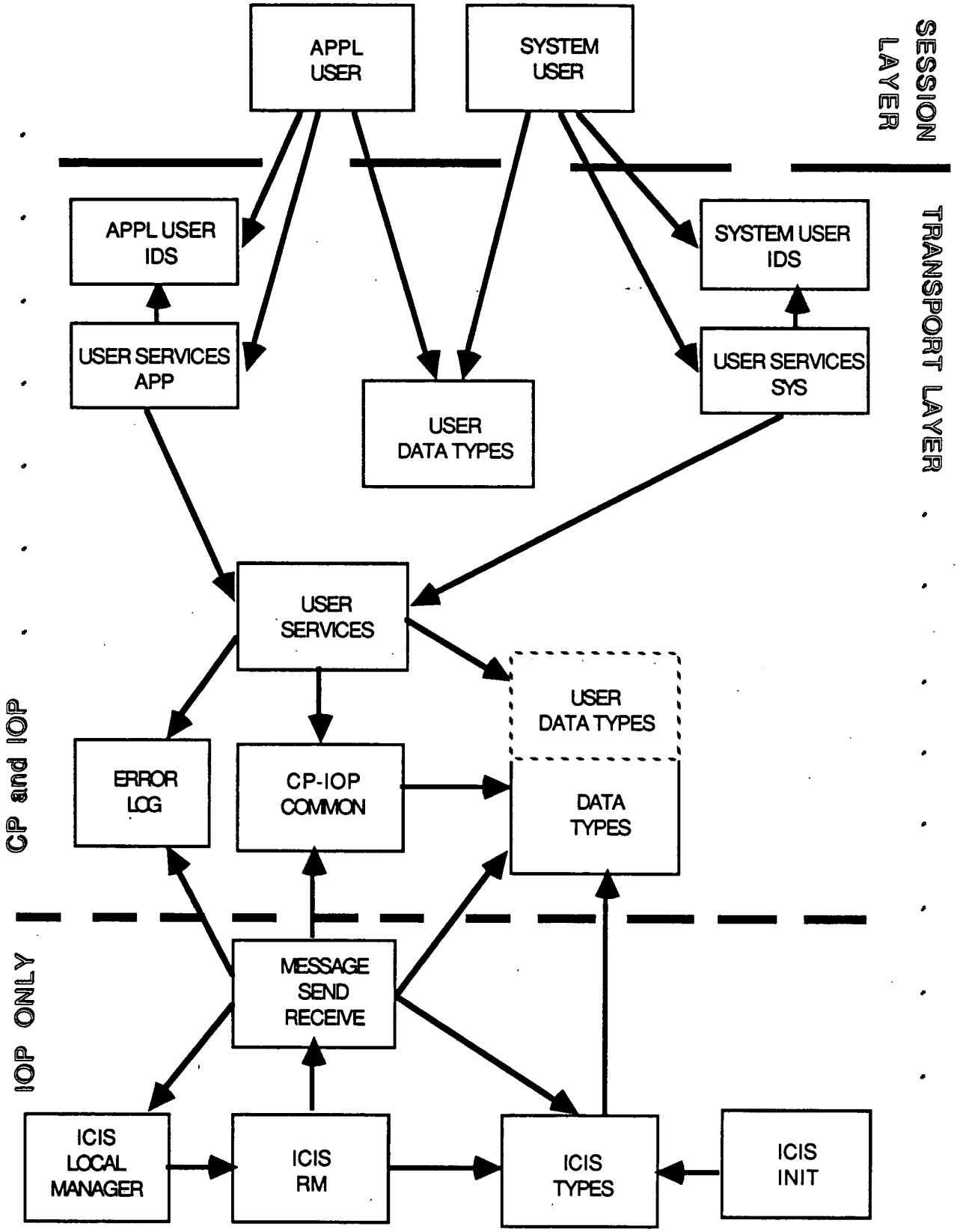


Figure 4-3. Transport Layer Packages

4.2.1.2 Data Structures

The User Services routines use the following data structures:

- User Output Buffer
- Output Buffer List
- User Input Buffer
- Input Buffer List
- CP / IOP Output Queue
- User Input Queues
- Input Event List
- Error Log
- CP and IOP Task Location Tables

All of these structures except for the Output Buffer List are shared with the Message Send-Receive task.

User Output Buffers. These buffers serve as a user's "OUT" box, where messages are held until they can be transmitted. The buffers are dynamically allocated in the User Services routine OUTPUT_SETUP, which the user process calls as part of its initialization procedure. The size of the buffers and the number allocated are specified by the user.

In addition to the actual data the user wants to send, the buffer contains control information used in transmitting the message:

- *Flag.* Current state of the buffer, i.e., whether it is unused, currently having an output message transferred into it, holding a message in the process of being sent, or holding a message which has been completely transmitted.
- *Message Priority.* A number from 1 to 100 which is assigned by the user.
- *Destination GPC.* A code identifying the destination GPC.
- *Destination User.* A code identifying the destination user.
- *Byte Count.* Length of the user message to be sent.
- *Error Code.* A code identifying the result of the transmission, e.g., message sent correctly, destination GPC not responding, etc.
- *User Error Check..* A flag indicating whether or not the user will later inquire about the status of the message transmission.
- *User Message ID.* A user-supplied 16-bit field used to identify the message when the user inquires about its transmission status.
- *User Check Time.* Latest time that status about the particular message will be held. If the user has not inquired about its status by this time, all information about the message will be deleted.

- *Outgoing Count.* Number of GPCs which have not yet acknowledged receiving the message. This field is used to determine how long to retain information about a broadcast message.

Output Buffer List. This list is a two-dimensional array which identifies the output buffers which have been allocated for all ICCS users. A user can allocate a maximum of 20 output buffers, implying that at most it can have 20 output messages in progress at one time.

User Input Buffers. These buffers serve as a user's "IN" box, where messages are held until a user requests them. The buffers are dynamically allocated in the User Services routine INPUT_SETUP, which the user process calls as part of its initialization procedure. The size of the buffers and the number allocated are specified by the user.

In addition to the actual data that is the user's message, the buffer contains control information:

- *Flag.* Current state of the buffer, i.e., whether it's empty, in the process of being filled, or contains a complete message ready for delivery to a user.
- *Message Priority.* A number from 1 to 100 which is assigned by the user.
- *Source GPC.* A code identifying the originating GPC.
- *Source User.* A code identifying the originating user.
- *Byte Count.* Length of the user message.

Input Buffer List. This list is a two-dimensional array which identifies input buffers which have been allocated for all IC users. A user process can allocate a maximum of 20 input buffers, implying that at most it can have 20 input messages pending.

CP / IOP Output Queue. The output queue is a one-dimensional array which identifies all buffers which currently have messages to be sent. The originating task is identified for each buffer, since there is only one queue per processor for all users. Output messages are sent from each queue in FIFO order; there is no provision for ordering according to the priority of the originating task or the priority of the individual message.

Each processor's queue is controlled by two indices: the add index and the process index. Since User Services and the Message Send-Receive task are operating on this queue asynchronously (User Services is adding entries; Message Send-Receive function is removing them), each uses its own index to maintain its current position in the array. The add index indicates the next available slot in the array where User Services can queue a new message. The process index indicates the next message in the queue to be transmitted by the Message Send-Receive function. When the process index is the same as the add index the queue is empty. In order for these indices to work correctly, the array can never be full. Since there is a maximum of 35 users, each of which could allocate 20 output buffers,

the theoretical maximum size of this queue is 700 entries. But given all the constraints of the system, it would be impossible to have this many output messages in progress at one time, and so only 50 entries have been allocated for each queue.

User Input Queue. This queue is a two-dimensional array which identifies for each user all buffers holding input messages. Input messages for any particular user are handed over in FIFO order; there is no provision for ordering them according to the priority of the sending task or the priority of the individual message. Each user's queue is controlled by two indices: the add index and the process index. Since the Message Send-Receive task and User Services are operating on this buffer asynchronously (Message Send-Receive is adding entries; User Services is removing them), each uses its own index to maintain its current position in the array. The add index indicates the next available slot in the array where Message Send-Receive can queue a new message. The process index indicates the next message in the queue to be given to a user via the User Services routine GET_INPUT. When the process index is the same as the add index the queue is empty. In order for these indices to work correctly, the array for any particular user can never be full. Since a user can allocate a maximum of 20 input buffers, each array has room for 21 entries.

The relationship between the various input and output buffers and queues is shown in Figure 4-4.

Input Event List. This list is a one-dimensional array which indicates for each user whether or not it should be signaled when an input message arrives. If the user does wish to be signaled, the list contains the address of the event to be used to start the user task.

Error Log. This structure is used by both User Services and the Message Send-Receive function to record unexpected events and error conditions. Refer to Section 4.2.2.1 for a complete description of this log.

CP and IOP Task Location Tables. These tables identify the processor ID, GPC ID, and Task Control Block for each ICCS user. Information about CP users is entered in the CP table; information about IOP users is entered in the IOP table.

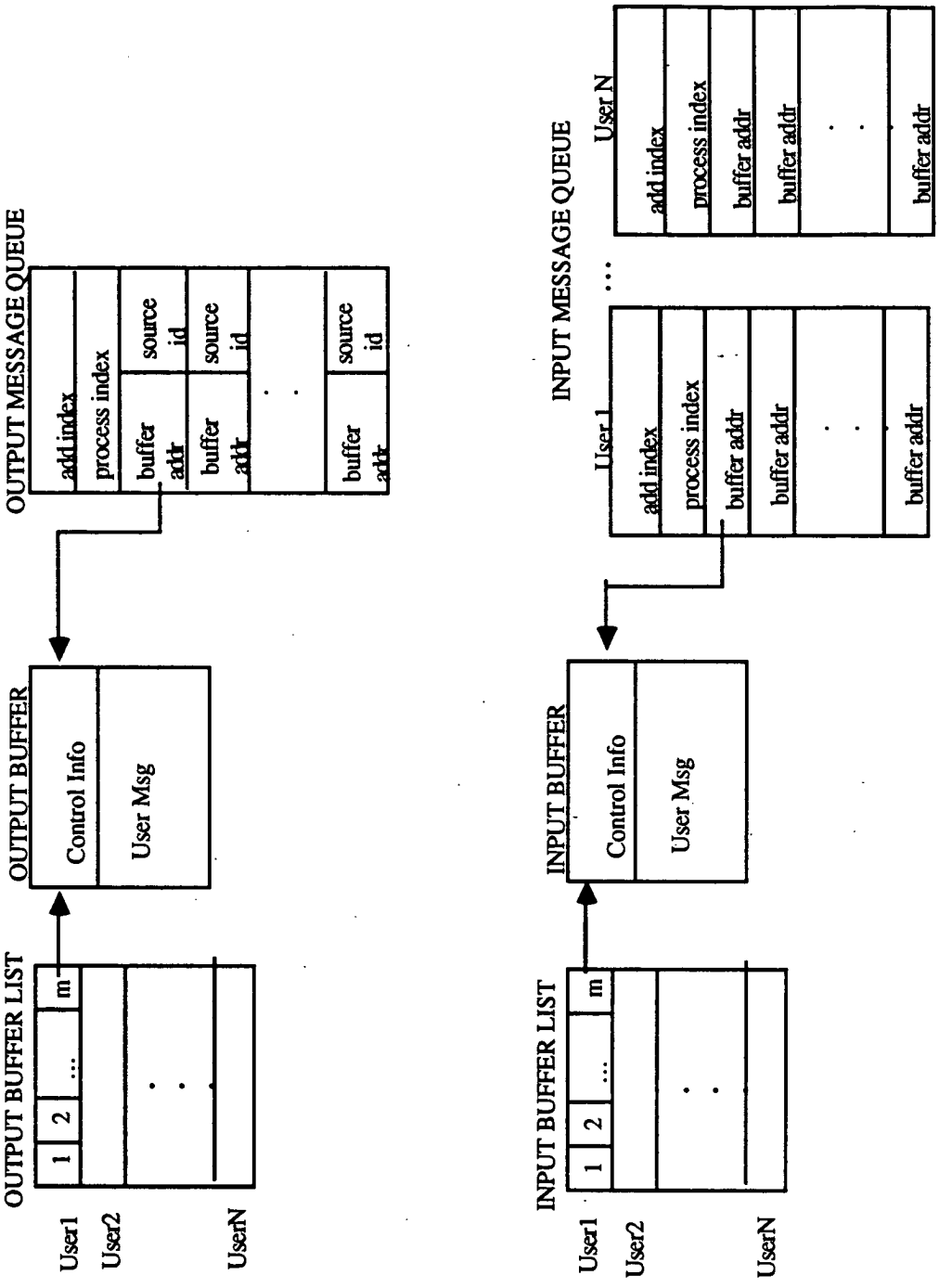


Figure 4-4. Input and Output Buffers and Queues

4.2.1.3 Process Descriptions

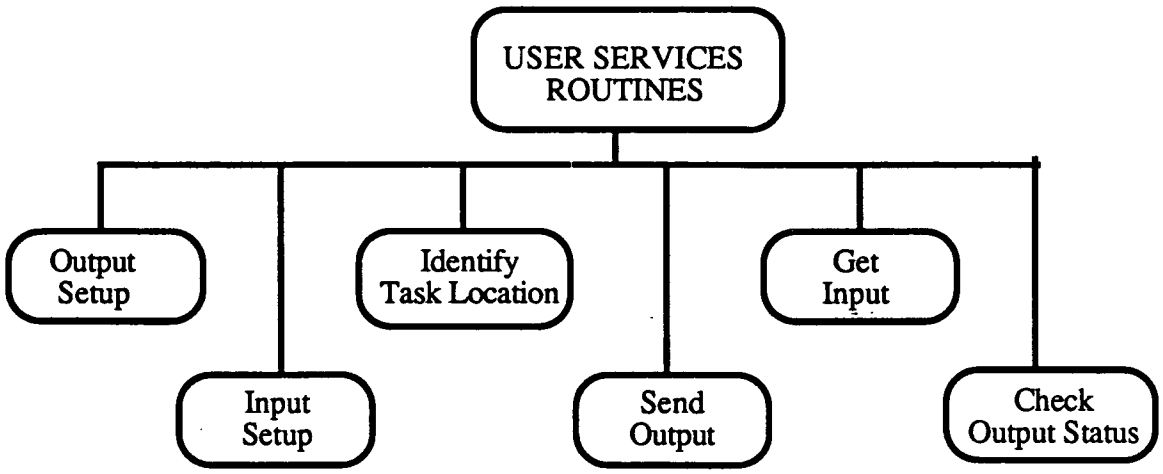


Figure 4-5. User Services

4.2.1.3.1 Process Name: OUTPUT_SETUP

Inputs:

- User ID
- Number and size of output buffers to be allocated

Outputs:

- Output buffers with state initialized to UNUSED
- CP/IOP output buffer lists

Description:

This routine is called by Transport Layer users during task initialization. Its function is to allocate buffers which will be used as temporary holding areas for messages the user wishes to send.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work.

The user has specified the number of buffers to be allocated and the length of the longest message. The actual buffer size is calculated by adding the size of the control information stored with each output message to the user's message length. Buffers for CP users are allocated from shared memory and their addresses stored in the CP Output Buffer List for the particular user. Buffers for IOP users are allocated from local memory and their addresses stored in the IOP Output Buffer List for the particular user. The flag field in each buffer allocated is initialized to UNUSED.

A user may allocate a maximum of 20 output buffers. If a user tries to allocate more than this number, 20 buffers will be allocated and an error will be recorded in the ICCS Error Log.

4.2.1.3.2 Process Name: INPUT_SETUP

Inputs:

- User ID
- Number and size of input buffers to be allocated

Outputs:

- Input buffers with state initialized to UNUSED
- CP/IOP Output Buffer Lists
- Event to trigger user task

Description:

This routine is called by Transport Layer users during task initialization. Its function is to allocate buffers to temporarily hold input messages that arrive for the user.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work.

The user has specified the number of buffers to be allocated and the length of the longest message expected to be received. The total buffer size is calculated by adding the size of the control information stored with each input message to the user's message length. Buffers for CP users are allocated from shared memory and their addresses stored in the CP Input Buffer List for the particular user. If a user wants to be signaled by an event when input arrives, the address of a previously allocated event is passed back so that the

user may schedule itself based on that event. Buffers for IOP users are allocated from local memory and their addresses stored in the IOP Input Buffer List for the particular user. As is the case for CP users, an IOP user who wants to be signaled by an event when input arrives is returned the address of a previously allocated event. The flag field in each buffer allocated is initialized to UNUSED.

A user may allocate a maximum of 20 input buffers. If a user tries to allocate more than this number, 20 buffers will be allocated and an error will be recorded in the ICCS Error Log.

4.2.1.3.3 IDENTIFY_TASK_LOCATION

Inputs:

- User ID
- GPC ID
- Processor ID
- Address of Task Control Block

Outputs:

- CP/IOP Task Location Tables

Description:

This routine is called by Transport Layer users during task initialization. Its function is to enter the initial location of each user into a central database so that its GPC, processor and task control block address are known globally.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work, which involves entering the GPC ID for the particular user into either the CP Task Location Table or IOP Task Location Table.

This routine is written as a function which returns a boolean, but the value being returned has no meaning. The routine was written in this way so that it could be invoked as part of the package specification which defined the user task, rather than having to be invoked in the package body. This, in turn, was a result of the design of the distributed system, in

which a given package body might reside on only one GPC but package specifications would reside on all GPCs.

4.2.1.3.4 SEND_OUTPUT

Inputs:

- User ID
- Address of message to be sent
- Destination GPC
- Destination user
- Message length
- Message priority
- User-specified message ID
- Flag for later error checking

Outputs:

- CP/IOP output message queues
- Selected output buffer
- Error code for caller

Description:

This routine is called by a Transport Layer user that wishes to send an output message. Since the Transport Layer must be independent of user-defined types, the message is specified by its starting address and size.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work.

The Transport Layer version finds a free holding buffer from among those the user has allocated at initialization and copies the message into it. It also enters all the control information required to send the message. Finally, it adds the buffer to the appropriate CP or IOP outgoing message queue. Since there is only one output queue per processor but multiple users, some strategy must be employed to ensure that the integrity of the queue is not compromised by users interrupting each other. This routine uses the lock feature provided by the operating system, so that the queue is locked before updates are begun and

unlocked when they are complete. Any task that attempts to lock the queue when it is already locked will be suspended until it is unlocked.

This routine returns to the caller any errors which prevented a message from being added to the output queue. Typical errors conditions are no free holding buffers, message size greater than that of the holding buffers, or IC network currently not in service. The error is also recorded in the ICCS Error Log.

4.2.1.3.5 GET_INPUT

Inputs:

- User ID
- Location for a message to be copied to
- User's input message queue

Outputs:

- IDs of originating user and GPC
- Message priority
- The message
- Input-available flag
- User's input message queue

Description:

This routine is called by a Transport Layer user that wants to determine if it has pending input messages and, if so, receive the oldest one. Since the Transport Layer must be independent of user-defined types, the message is specified by its starting address and size.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work.

The Transport Layer version checks the user's input queue for messages and sets a flag indicating whether or not there are messages pending. If messages are pending, it copies the oldest one to the user's area and also returns the ID of the originating user, the ID of the originating GPC, and the message priority. Finally, the message is removed from the input queue.

4.2.1.3.6 CHECK_OUTPUT_STATUS

Inputs:

- User ID
- User message ID

Outputs:

- Current status of the specified output message

Description:

This routine is called by a Transport Layer user that wants to check on the current status of a particular output message. The status information returned indicates whether the transmission is still in progress or has completed, and if completed, whether successfully or with a particular type of error.

Versions of this routine exist in three packages:

USER_SERVICES_SYS	(for system users)
USER_SERVICES_APP	(for application users)
USER_SERVICES	(Transport Layer version)

The purpose of the version called directly by a user is to translate the enumeration-type user ID known to the user into the integer-type user ID known to the Transport Layer. It then calls the Transport Layer version of the routine to do the actual work.

The user's output buffers are searched to find a match with the user's message ID field. If no match is found, a status of MSG_ID_NOT_FOUND is returned. Otherwise, the current status from the buffer is returned. It will be one of the following:

NO_ERRORS
MSG_NOT_COMPLETE
TRANSMIT_ERROR
DEST_GPC_NOT_RESPONDING
DEST_GPC_CANT_DELIV

4.2.2 Software Specifications: Message Send-Receive Task

4.2.2.1 Data Structures

The Message Send-Receive task uses the following data structures:

- Basic Packet
- Output Packet
- Input Packet
- Message Status Block
- Message Status Block List
- Pending Message List
- Outgoing Message Counts
- Message Send-Receive Input Buffer
- Error Log

The Message Send-Receive Input Buffer is shared with the ICIS Redundancy Management function; the other structures are internal to the Message Send-Receive task.

Basic Packet. The maximum data (packet) size that can be handled by the ICIS during a transmit or receive operation is 127 bytes. Not only must a user message be divided into sections if it is bigger than this maximum size, but 12 bytes of control information must be attached to each message (or section of a message) so that the message can be identified to the Message Send-Receive task at its destination. After allowing for additional control information required by the hardware for transmission and status information which is appended by the hardware when a message is received, the maximum amount of user data that can be included in one packet is 104 bytes. The format of the basic packet used by Message Send-Receive is shown in Figure 4-6.

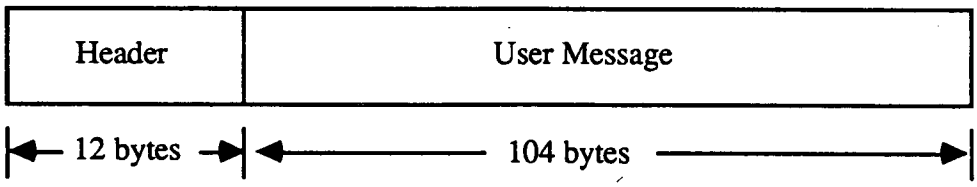


Figure 4-6. Basic Packet

The packet header contains the following information:

- *Packet type*. A code indicating whether the packet is the first packet of a message (MSG), a continuation packet (MSG_CONT), an acknowledgement (ACK), or an acknowledgement for a packet which could not be delivered to its end user (UNDELIV).
- *Packet number*. Sequence number of the packet within the message.

- *Total packets.* Total number of packets in this message.
- *Message priority.* The user-assigned priority for this message.
- *Source GPC.* A code identifying the originating GPC.
- *Source user.* A code identifying the originating user.
- *Destination GPC.* A code identifying the destination GPC.
- *Destination user.* A code identifying the destination user.
- *Message ID.* A 32-bit field that uniquely identifies the message when the source and destination ID fields are the same.

The contents of a packet vary according to the packet type. ACK and UNDELIV packets contain only the header information. The MSG packet is used to send the first (or only) packet of a message and contains the header plus up to 104 bytes of user data. The MSG_CONT packet is used to send continuation packets of messages longer than 104 bytes. It also contains the header plus up to 104 bytes of user data.

Output Packet. Additional information must be inserted at the front of a packet before it can be transmitted. The first two bytes are required by the ICIS hardware, and the next two bytes are used by the ICIS Redundancy Management function on the receiving GPC. The format of the output packet transmitted by the Message Send-Receive task is shown in Figure 4-7.

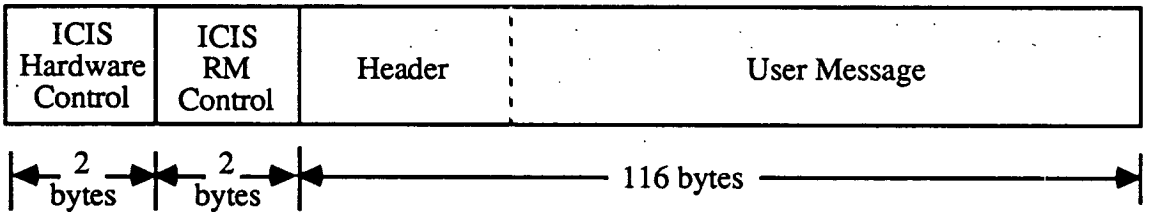


Figure 4-7. Output Packet

The two control bytes required by the ICIS hardware are:

- *Byte count.* Number of bytes to be transmitted in this packet. This count includes all data after the byte count field. It is expressed in the form 128 - n, where n is the number of bytes.
- *Address of destination GPC.* A code to which the destination GPC will respond when it sees the message on the network. This code uses the same values as the destination GPC code mentioned elsewhere in this chapter.

The control information required by the receiving ICIS Redundancy Management function is:

- *Sender redundancy.* A code identifying which layers the sending GPC is transmitting on. This information is used by the ICIS Redundancy Management in its fault detection function.

Input Packet. An incoming packet has certain status information attached to it by the ICIS hardware before it is stored in the ICIS dual-ported memory. The ICIS Redundancy Management function removes most of this information but extracts selected pieces before presenting a packet to Message Send-Receive. The format of the input packet available to Message Send-Receive is shown in Figure 4-8.

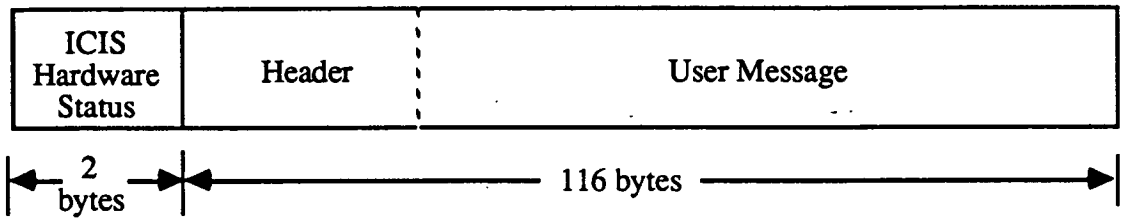


Figure 4-8. Input Packet

The two bytes of ICIS hardware information include the following:

- *Byte count.* Number of bytes in this packet. This count does not include the 2 bytes of ICIS hardware information.
- *Time.* Time the input packet was read into the ICIS dual-ported memory. This value is derived from the fault-tolerant clock and has a granularity of 66 microseconds.

Message Status Block. The Message Send-Receive task keeps information about each ongoing message in a Message Status Block. This structure is allocated dynamically when a message first appears and deallocated when all handling of the message is complete. The Message Status Block enables the Message Send-Receive task to match an incoming acknowledgement with the source message, to control the sending of multi-packet output messages, and to assemble multi-packet input messages. Since several messages may be in progress at any one time, Message Status Blocks are connected to each other in a linked list.

A Message Status Block contains the following information.

- *Source GPC.* A code identifying the originating GPC.
- *Source User.* A code identifying the originating user.
- *Destination GPC.* A code identifying the destination GPC.
- *Destination User.* A code identifying the destination user.
- *Message ID.* A 32-bit field that uniquely identifies the message when the source and destination id fields are the same.
- *Packet send time.* The time the packet was transmitted. This field is used to determine if a message has timed out, i.e., not been acknowledged within the allowable time.

- *Message status.* Current status of the message, i.e., whether it is an output message waiting for an acknowledgement, an multi-packet input message waiting for continuation packets, or an output message on the pending list.
- *Output buffer pointer.* Address of the holding buffer for an output message.
- *Output data.* Address of the user data within the holding output buffer.
- *Input buffer pointer.* Address of the holding buffer for an input message.
- *Input data.* Address of the user data within the holding input buffer.
- *Last packet byte count.* Number of bytes to be transmitted in the last packet of a multi-packet output message.
- *Total packets.* Total number of packets in a multi-packet message.
- *Current packet being transmitted.* Number of the packet currently being sent in a multi-packet output message.
- *Total packets received.* Number of packets received so far of a multi-packet input message.
- *Last packet received.* Number of the most recently received packet in a multi-packet input message.
- *Previous Message Status Block.* A pointer to the previous Message Status Block in the linked list.
- *Next Message Status Block.* A pointer to the next Message Status Block in the linked list.

Message Status Block List. This list identifies all messages currently in progress on the particular GPC. Initial entry into the list is made through an array indexed by user ID (source user for output messages, destination user for input messages). All Message Status Blocks for a particular user are then joined in a linked list. The Message Status Block List is illustrated in Figure 4-9.

Pending Messages List. As described in Section 4.1.2 on the functional design of the Message Send-Receive task, messages are sent using a protocol which requires that each packet transmitted must be explicitly acknowledged and that the number of packets allowed to be unacknowledged at any one time is limited. When a message is to be sent to a GPC already receiving its maximum messages, the latest message is put on the pending list and will be transmitted as soon as an acknowledgement to an outstanding packet arrives. A separate pending list is maintained for each destination GPC.

The Pending Messages List is illustrated in Figure 4-10. Each GPC's list is a one-dimensional array of pointers to Message Status Blocks and is controlled by two indices: the add index and the process index. Since two different processes are operating on this list asynchronously (one process to add messages, another process to remove them), each

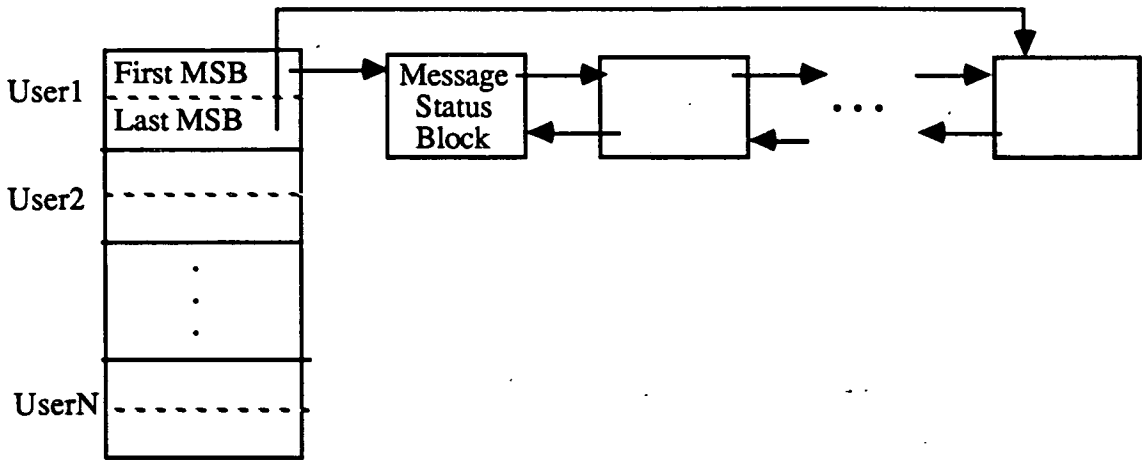


Figure 4-9. Message Status Block List

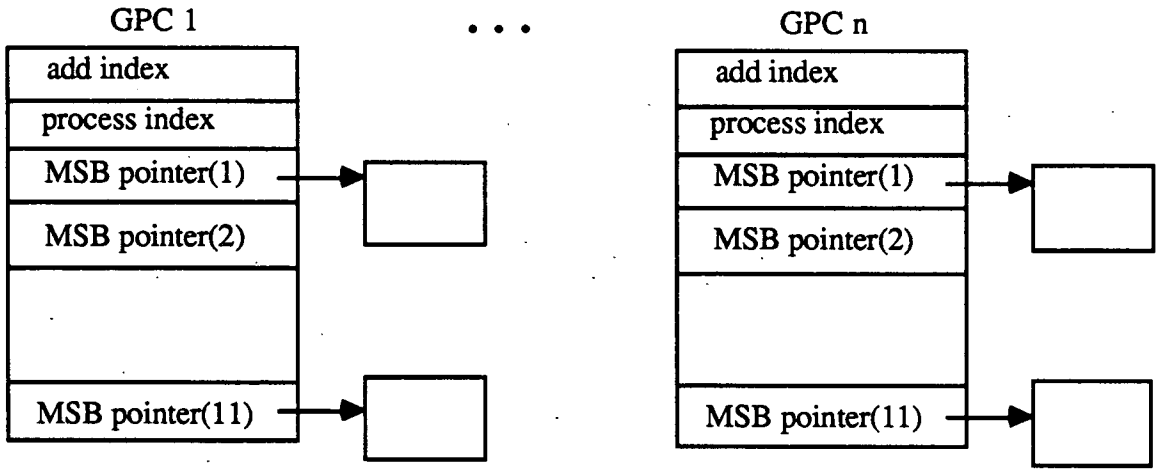


Figure 4-10. Pending Messages List

process uses its own index to maintain its current position in the array. The add index indicates the next available slot where a new message can be added; the process index indicates the next message to be sent and then removed from the array. When the add index is the same as the process index, the list is empty. In order for these indices to work correctly, the array can never be full. An arbitrary maximum of 10 pending messages per GPC has been established; therefore each Pending Message List has room for 11 entries.

Outgoing Message Counts. The number of output messages currently being sent to each GPC is maintained in a one-dimensional array indexed by GPC ID. For programming convenience, a total count is also maintained as an integer variable.

4.2.2.2 Process Descriptions

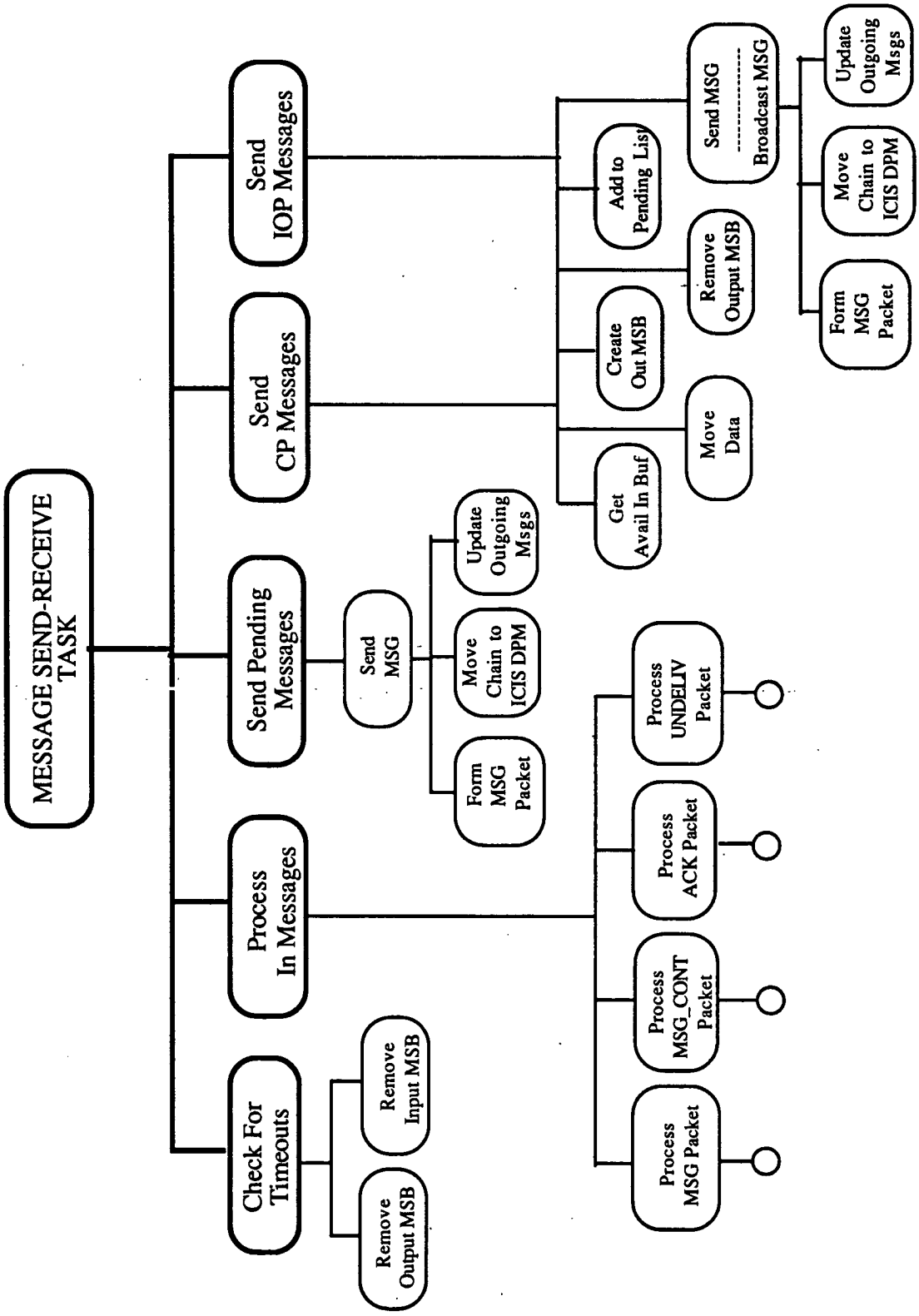


Figure 4-11. Message Send-Receive Task

Figure 4-11. Message Send-Receive Task (cont.)

Message Send-Receive Input Buffer. This data structure is used by the ICIS Redundancy Management task to pass to the Message Send-Receive task the congruent packets which it has formed from the multiple copies available in the ICIS dual-ported memory.

This structure is a one-dimensional array which is controlled by two indices: the FDIR index and the MSR index. Since ICIS Redundancy Management and Message Send-Receive are operating on this buffer asynchronously (ICIS Redundancy Management is adding packets; Message Send-Receive is removing them), each uses its own index to maintain its current position in the array. The FDIR index indicates the next available slot in the array where ICIS Redundancy Management can add a new packet. The MSR index indicates the next packet to be removed from the array and distributed to a user. When the MSR index is the same as the FDIR index, the array is empty. In order for these indices to work correctly, the array can never be full. Since 36 input buffers have been allocated in the ICIS dual-ported memory, this array has room for 37 entries.

Error Log. This structure is used by both User Services and the Message Send-Receive function to record unexpected events and error conditions. Each entry in the log contains an identifier for the routine logging the entry, a code identifying the error condition, the user and GPC involved, and the time of day. There is one log on each processor. Only User Services makes entries in the CP log, while both User Services and Message Send-Receive make entries in the IOP log.

4.2.2.2.1 MESSAGE SEND-RECEIVE Task Body

Inputs:

- Count of ongoing messages
- Count of pending messages
- Indices of Message Send-Receive input buffer
- Indices of CP output queue
- Indices of IOP output queue

Outputs:

- None

Description:

The Message Send-Receive task has two main functions: (1) to transmit output messages as requested by Transport Layer users, and (2) to distribute incoming messages. It has two additional functions which are offshoots of these primary ones: (1) to check for packets which have not been responded to within the time limit, and (2) to determine if messages on the pending queue can be sent. Accordingly, the subtasks of Message Send-Receive are:

(1) check for timeouts
(2) process incoming packets
(3) check pending messages
(4) send output messages

The Message Send-Receive task is started by an event, which can be set either by the ICIS Redundancy Management when there are input packets to process or by User Services when a user has a message to send. Every time it is scheduled, however, it checks its entire list of possible things to be done. Furthermore, as long as it has handled either an input message or a new output message during any particular iteration, it continues to check. This strategy allows multiple processors (CP, IOP) to be signaling the task simultaneously and allows Message Send-Receive to handle all requests as promptly as possible with a minimum of context switching. The Message Send-Receive task thus operates asynchronously with the processes it is serving.

The order in which the subtasks are done is important. Timeouts must be checked first so that resources used by any timed-out messages can be freed. Input packets need to be checked next so that acknowledgments can free resources being used for output messages. Pending messages must be sent before newly queued output messages.

Normally the setting of an event when a task is not at its Wait_For_Schedule causes the event to be lost. The Message Send-Receive task uses a special parameter provided by the SCHEDULE routine so that this does not happen. This task will always be scheduled after its event was set, even if it was not previously at its Wait_For_Schedule point.

4.2.2.2.2 CHECK_FOR_TIMEOUTS

Inputs:

- Message Status Block list

Outputs:

- Count of ongoing messages
- Status fields in output buffers

Description:

Messages in progress might time out in one of two ways. An output message could time out because an acknowledgment was not received, either for the first packet in a message or for continuation packets in a multi-packet message. A multi-packet input message could time out because an expected continuation packet was not received. This routine compares the current time to the time the outgoing packet in question (MSG, MSG_CONT for output messages, ACK for input messages) was sent. A time limit of 400 milliseconds is allowed for the expected packet to arrive. After this time, the Message Status Block is deleted. For output messages, the error code field in the user's output buffer is set to DEST_GPC_NOT_RESPONDING. For an input message, the user's input buffer is freed.

4.2.2.2.3 PROCESS_IN_MESSAGES

Inputs:

- Indices of Message Send-Receive input buffer
- Next available packet from Message Send-Receive input buffer

Outputs:

- MSR_index of Message Send-Receive input buffer

Description:

This routine determines if there are input packets to be processed by comparing the MSR_index to the FDIR_index in the Message Send-Receive input buffer. There are packets to be processed whenever the two indices are not equal. One of four subroutines is called to process the packet, depending on the packet type.

4.2.2.2.4 SEND_PENDING_MESSAGES

Inputs:

- Outgoing message count for each GPC
- Indices of Pending Message List for each GPC

Outputs:

- Process_index of Pending Message List for each GPC
- Count of pending messages

Description:

This routine is invoked when there is at least one pending output message. It checks the current output message count and the Pending Message List for each GPC to determine which pending messages can be sent.

4.2.2.2.5 SEND_CP_MESSAGES

Inputs:

- Indices of CP output queue
- Next entry from CP output queue
- CP and IOP Task Location Tables
- Outgoing message count for each GPC

Outputs:

- Process_index of CP output queue
- Local destination user's input buffer and input queue

Description:

This routine sends messages as requested by CP users by sequentially processing entries in the CP output queue. The steps required to send a message vary according to the destination GPC, which will fall into one of three categories:

- Local GPC (i.e., destination and source GPCs are the same)
- All GPCs
- Single remote GPC

Local GPC. It may be that the destination user of a particular message resides on the same GPC as the sender. This would be the case, for example, where a migratable function currently is executing on the same GPC as a task with which it communicates, or where a message must be sent to some user residing on all GPCs, including the one sending the message. In this case, the IC network is not used; rather the message is moved directly into one of the destination user's holding input buffers and added to its input queue. If no input buffers are available to hold the message, an error is recorded in the ICCS Error Log.

All GPCs. When a message is to be sent to all GPCs in the system, the preferred method is to transmit it in broadcast mode (i.e., addressed to all GPCs). Although only one output packet needs to be formed and one ICIS chain executed, a Message Status Block must be created for every destination GPC. This is because the receiving GPCs will be acknowledging the message asynchronously and, if it is a multi-packet message, receiving continuation packets asynchronously.

The message cannot be transmitted in broadcast mode if any of the other GPCs is currently receiving its maximum messages from the sending GPC. In this case the message is put on the pending queue for the GPC currently at its maximum and is sent individually to the other GPCs.

Single Remote GPC. A Message Status Block is created for the destination GPC. If the GPC is currently receiving its maximum messages from the sending GPC, the message is put on the pending queue; otherwise the packet is sent immediately.

The routine that is invoked to format and transmit the message returns a boolean indicating whether or not the packet was successfully transmitted. If it was not, the Message Status Block(s) is immediately deleted.

This procedure concludes by updating the process index of the CP output queue.

4.2.2.2.6 SEND_IOP_MESSAGES

Inputs:

- Indices of IOP output queue
- Next entry from IOP output queue
- CP and IOP Task Location Tables
- Outgoing message count for each GPC

Outputs:

- Process_index of IOP output queue
- Local destination user's input buffer and input queue

Description:

This routine sends messages as requested by IOP users by sequentially processing entries in the IOP output queue. The steps required are exactly the same as those required to send a CP message except that the IOP rather than the CP output queue is used. The reader should refer to the previous section for details.

4.2.2.2.7 FORM_MSG_PACKET

Inputs:

- Message Status Block
- Holding buffer containing the message to be sent

Outputs:

- MSG output packet

Description:

This routine forms a MSG packet for a message that is to be transmitted. The MSG packet is used to send the first 104 bytes of any message; if the total message is shorter than this, only the number of bytes in the total message is sent. The necessary control information is inserted at the front of the packet.

4.2.2.2.8 FORM_ACK_PACKET

Inputs:

- Current packet from Message Send-Receive input buffer

Outputs:

- ACK output packet

Description:

This routine formats an ACK for the MSG or MSG_CONT packet currently being processed from the Message Send-Receive input buffer. An ACK packet contains only the 12 bytes of control information needed to identify the message back at its originating GPC.

4.2.2.2.9 FORM_UNDELIV_PACKET

Inputs:

- Current packet from Message Send-Receive input buffer

Outputs:

- UNDELIV output packet

Description:

This routine formats an UNDELIV packet for the MSG packet currently being processed from the Message Send-Receive input buffer. An UNDELIV packet contains only the 12 bytes of control information needed to identify the message back at its originating GPC.

4.2.2.2.10 FORM_MSG_CONT_PACKET

Inputs:

- Message Status Block
- Holding buffer containing the message to be sent

Outputs:

- MSG_CONT output packet

Description:

This routine forms MSG_CONT packets for messages that are too long to be transmitted in one packet. The last MSG_CONT packet for the message will contain only the number of bytes in the message not yet sent; otherwise it will contain 104 bytes of message data. The necessary control information is inserted at the front of the packet.

4.2.2.2.11 MOVE_CHAIN_TO_ICIS_DPM

Inputs:

- Output chain to be used for transmission

Outputs:

- Solicited chain in ICIS DPM

Description:

This routine copies the chain to be used for ICIS transmission from local RAM to the ICIS dual-ported memory. This chain is defined as a constant and is copied to the ICIS dual-ported memory before every transmission.

4.2.2.2.12 SEND_MSG

Inputs:

- Message Status Block for the message

Outputs:

- MSG output packet in ICIS dual-ported memory
- Status of transmission

Description:

This routine transmits a MSG packet. The packet to be transmitted and the solicited chain to be used are moved to ICIS dual-ported memory. Then a subroutine provided by the ICIS_LOCAL_MANAGER is called to check that the ICIS is ready to transmit and to start the output chain. The status field in the Message Status Block is set to AWAITING_ACK, and the packet send time field is set to the current time. Finally, the count variables for outgoing messages are updated.

If the output chain could not be started or did not complete successfully, an error is recorded in the ICCS Error Log and an error indication is returned to the caller.

4.2.2.2.13 SEND_MSG_CONT

Inputs:

- Message Status Block for the message

Outputs:

- MSG_CONT output packet in ICIS dual-ported memory
- Status of transmission

Description:

This routine transmits a MSG_CONT packet. The packet to be transmitted and the solicited chain to be used are moved to ICIS dual-ported memory. Then a subroutine provided by the ICIS_LOCAL_MANAGER is called to check that the ICIS is ready to transmit and to start the output chain. The status field in the Message Status Block is set to AWAITING_ACK, and the packet send time field is set to the current time. Finally, the count variable in the output buffer is updated.

If the output chain could not be started or did not complete successfully, an error is recorded in the ICCS Error Log and an error indication returned to the caller.

4.2.2.2.14 SEND_ACK

Inputs:

- Message Status Block for the message

Outputs:

- ACK output packet in ICIS dual-ported memory

Description:

This routine transmits an ACK packet. The packet to be transmitted and the solicited chain to be used are moved to ICIS dual-ported memory. Then a subroutine provided by the ICIS_LOCAL_MANAGER is called to check that the ICIS is ready to transmit and to start the output chain. The status field in the Message Status Block is set to AWAITING_MSG_CONT, and the packet send time field is set to the current time.

If the output chain could not be started or did not complete successfully, an error is recorded in the ICCS Error Log.

4.2.2.2.15 SEND_UNDELIV

Inputs:

- None

Outputs:

- UNDELIV output packet in ICIS dual-ported memory

Description:

This routine transmits an UNDELIV packet. The packet to be transmitted and the solicited chain to be used are moved to ICIS dual-ported memory. Then a subroutine provided by the ICIS_LOCAL_MANAGER is called to check that the ICIS is ready to transmit and to start the output chain. If the output chain could not be started or did not complete successfully, an error is recorded in the ICCS Error Log.

4.2.2.2.16 BROADCAST_MSG

Inputs:

- Message Status Blocks for the message

Outputs:

- MSG output packet in ICIS dual-ported memory
- Status of transmission

Description:

This routine broadcasts a MSG packet. The packet to be transmitted and the solicited chain to be used are moved to ICIS dual-ported memory. The destination address at the beginning of the output packet is set to address all GPCs. Then a subroutine provided by the ICIS_LOCAL_MANAGER is called to check that the ICIS is ready to transmit and to start the output chain.

For a broadcast message, a Message Status Block has previously been created for each receiving GPC. This routine sets the status field in each Message Status Block to AWAITING_ACK and sets the packet send time field to the current time. Then the count variables for outgoing messages are updated.

If the output chain could not be started or did not complete successfully, an error is recorded in the ICCS Error Log and an error indication is returned to the caller.

4.2.2.2.17 UPDATE_OUTGOING_MSGS

Inputs:

- Increment value
- Destination GPC

Outputs:

- Outgoing message count for the destination GPC
- Total outgoing message count

Description:

This routine updates two variables which contain the number of outgoing messages in progress on the particular GPC. One variable is an array which identifies for each GPC the number of outgoing messages addressed to that GPC. The other variable contains the total number of outgoing messages currently in progress.

4.2.2.2.18 GET_AVAIL_IN_BUF

Inputs:

- Processor ID of the destination user
- Destination user
- Holding input buffer list for that user

Outputs:

- Flag field in the selected holding buffer

Description:

This routine scans all of the holding input buffers allocated for a particular user until it finds an empty one. It sets the flag in that buffer to BEING_FILLED and returns the buffer address to the calling routine. If there are no empty buffers, a null address is returned.

4.2.2.2.19 MOVE_DATA

Inputs:

- Source address
- Destination address
- Number of bytes to be copied

Outputs:

- Area pointed to by the destination address

Description:

This is an assembly language routine that copies data, for the length specified, from one memory location to another. Its purpose is to copy data in the most efficient way possible and in a way independent of particular data types.

4.2.2.2.20 CREATE_OUT_MSB

Inputs:

- Current message from CP or IOP output message queue
- Destination GPC
- Message ID

Outputs:

- Message Status Block
- Message Status Block List for the source user

Description:

This routine dynamically allocates a Message Status Block for the output message currently being processed, initializes its fields, and adds it to the list of Message Status Blocks for the task originating the message.

4.2.2.2.21 CREATE_IN_MSB

Inputs:

- Current packet from the Message Send-Receive input buffer
- Address of holding buffer for this packet

Outputs:

- Message Status Block
- Message Status Block List for the destination user

Description:

This routine dynamically allocates a Message Status Block for the input message currently being processed from the Message Send-Receive input buffer. It initializes the fields in the Message Status Block and adds it to the list of Message Status Blocks for the destination user. Note that this routine is only called for multi-packet input messages. For single-packet messages a temporary Message Status Block is created (see following section).

4.2.2.2.22 CREATE_TEMP_MSB

Inputs:

- Current packet from the Message Send-Receive input buffer
- Address of holding buffer for this packet

Outputs:

- Message Status Block

Description:

This routine dynamically allocates a Message Status Block for the input message currently being processed from the Message Send-Receive input buffer and initializes the fields. This routine is only called when an input message is contained in a single packet, and the Message Status Block is not added to the list of Message Status Blocks for the destination user.

4.2.2.2.23 FIND_OUTPUT_MSB

Inputs:

- Source GPC
- Source user
- Destination GPC
- Destination user
- Message ID

Outputs:

- Address of Message Status Block

Description:

This routine searches the Message Status Block list for the source user for a match on the input parameters listed above. If a match is found, the address of the Message Status Block is returned to the caller; otherwise a null address is returned.

4.2.2.2.24 FIND_INPUT_MSB

Inputs:

- Destination GPC
- Destination user
- Source GPC
- Source user
- Message ID

Outputs:

- Address of the Message Status Block

Description:

This routine searches the Message Status Block list for the destination user for a match on the input parameters listed above. If a match is found, the address of the Message Status Block is returned to the caller; otherwise a null address is returned.

4.2.2.2.25 REMOVE_INPUT_MSB

Inputs:

- Message Status Block

Outputs:

- Message Status Block List for source user

Description:

This routine removes the specified Message Status Block from the linked list of Message Status Blocks for the destination user. It then dynamically deallocates the memory used by the Message Status Block. If the Message Status Block cannot be found in the linked list, an error is recorded in the ICCS Error Log.

4.2.2.2.26 REMOVE_OUTPUT_MSB

Inputs:

- Message Status Block

Outputs:

- Message Status Block for destination user

Description:

This routine removes the specified Message Status Block from the linked list of Message Status Blocks for the source user. It then dynamically deallocates the memory used by the Message Status Block. If the Message Status Block cannot be found in the linked list, an error is recorded in the ICCS Error Log.

4.2.2.2.27 ADD_TO_PENDING_LIST

Inputs:

- Message Status Block for the message to be added

Outputs:

- List of pending messages for the appropriate GPC
- Status field in Message Status Block
- Count of pending messages

Description:

This routine adds to the pending list a message which cannot currently be sent because the destination GPC is already receiving its maximum number of messages. There is a separate pending list for each GPC. The new message is added in the slot pointed to by the add_index; this index is then updated to point to the next slot.

A maximum of 10 messages may be pending for any GPC. If the pending list for a particular GPC is full, the new message is not added and an error is recorded in the ICCS Error Log.

4.2.2.2.28 SIGNAL_INPUT

Inputs:

- Destination user
- CP, IOP Task Location Tables
- CP, IOP Signal Arrays
- CP, IOP Event Arrays

Outputs:

- Destination user's event

Description:

This routine is called after a complete input message has been assembled for a user. If the particular user has specified that it is to be signaled (i.e., scheduled by event) when an input message arrives, this routine causes the appropriate event to be set.

4.2.2.2.29 UPDATE_IN_MESSAGES

Inputs:

- Message Status Block
- CP, IOP Task Location Tables
- CP, IOP Input Queues

Outputs:

- Input Queue for the destination user

Description:

This routine adds the message referenced by the Message Status Block to the input queue for the destination user. The message is added in the slot indicated by the queue's add index; then the add index is updated to point to the next slot.

4.2.2.2.30 PROCESS_MSG_PACKET

Inputs:

- Current packet from the Message Send-Receive input buffer
- CP and IOP Task Location Tables

Outputs:

- Message Status Block
- Holding buffer containing the input message

Description:

This routine processes an incoming MSG packet, which is the first (and possibly only) packet of any message. It uses the destination task ID from the packet to look up in the Task Location Tables whether the message should be delivered to the CP or IOP. Then it gets a free holding buffer from among those allocated by the user. If no holding buffers are available, the message is not deliverable, and an UNDELIV packet is returned to the originating GPC. In addition, an error is recorded in the ICCS Error Log. If a holding buffer is available, a Message Status Block is created to keep track of the message. In the case of a message which fits completely into the MSG packet, the Message Status Block is only temporary and will be deleted at the end of the routine. The message data is then moved to the user's holding buffer.

If the entire message is contained in this one packet, the message may now be added to the user's input queue and the user signaled (if he has previously specified that this should occur). If more packets are expected, variables are updated so that the current position in the message is correctly maintained. In either case, an ACK is sent to the originating GPC.

4.2.2.2.31 PROCESS_MSG_CONT_PACKET

Inputs:

- Current packet from the Message Send-Receive input buffer
- Message Status Block previously created for the message

Outputs:

- Message Status Block
- Holding buffer containing the input message

Description:

This routine adds a continuation packet to the packets previously received in a multi-packet message, so that the complete message is eventually reassembled at the receiving site.

First the Message Status Block previously created for the message is located. The input packet is then stripped of its control data and moved to the current position in the holding buffer. The fields in the Message Status Block which are maintaining information about how much of the message has been received are updated to reflect the new packet., and an ACK is sent to the originating GPC.

If the message is now completely assembled (i.e., this packet is the last one), the message may now be added to the user's input queue and the user signaled (if he has previously specified that this should occur).

If the Message Status Block created for the message cannot be found or a packet arrives out of sequence, an error is recorded in the ICCS Error Log.

4.2.2.2.32 PROCESS_ACK_PACKET

Inputs:

- Message Status Block previously created for the message
- Holding buffer for the message

Outputs:

- Error-code and flag fields in the holding buffer
- Outgoing message counts

Description:

This routine processes an ACK that is received for a previously transmitted packet.

First the Message Status Block previously created for the message is located. If the message could fit into one packet, the transmission is now complete, so the error-code field in the holding buffer is set to NO_ERRORS and the flag field is set to MSG_SENT. The

Message Status Block is deleted, and the counts of outgoing messages are updated. If the message could not fit into one packet, a continuation packet must now be sent.

If the Message Status Block previously created for the message cannot be found, an error is recorded in the ICCS Error Log.

4.2.2.2.33 PROCESS_UNDELIV_PACKET

Inputs:

- Message Status Block previously created for the message
- Holding buffer for the message

Outputs:

- Error-code and flag fields in the holding buffer
- Outgoing message counts

Description:

This routine processes an UNDELIV that is received for a previously transmitted packet. The transmitted packet could not be delivered to the end user because there were no free holding buffers. Note that an UNDELIV may occur only in response to the first packet of a message.

The UNDELIV indication is recorded in the ICCS Error Log. The error-code field in the holding buffer is set to DEST_GPC_CANT_DELIV and the flag field is set to MSG_SENT. The Message Status Block is deleted, and the counts of outgoing messages are updated.

4.2.2.2.34 LOG_IC_ERROR

Inputs:

- Error identification
- Error location within Message Send-Receive task
- Source or destination user for the message being processed
- Source or destination GPC for the message being processed

Outputs:

- Entry in error log
- Error log index

Description:

This routine creates a new entry in the ICCS Error Log. It fills in the error identification, error location, user and GPC fields from the input parameters. It gets the current date and time from the standard "CLOCK" routine. It then updates the log index to point to the next available slot.

4.2.2.2.35 DISPLAY_IC_ERROR_LOG

Inputs:

- ICCS Error Log

Outputs:

- Formatted CRT display

Description:

This routine translates the codes contained in each entry in the ICCS Error Log into character strings using the Ada 'image attribute. It formats each entry in the log for display on a VT220 screen.

4.2.3 Software Specifications: ICIS Redundancy Management

The software objects used to implement the ICIS Redundancy Management functions are shown in Figure 4-12. The software architecture for these functions is driven by real-time system requirements. The implementation of the functions related to the processing of data received from the IC network, both the Source Congruency function and the FDI function related to received data, must take into consideration the functions' impact on the time lag being introduced into the inter-computer communications process. These data reception functions are separated into a single task, the ICIS Redundancy Management (ICIS_RM) task, which is scheduled whenever new input has arrived. Since the ICIS hardware itself does not generate an interrupt when input is received, a dedicated interval timer is set to go off every 5 milliseconds, and the interrupt handler checks if new ICIS input has arrived. The ICIS_RM task then performs the Source Congruency and FDI functions on a packet-by-packet basis until all of the new input packets have been processed. A single congruent copy of each received packet is passed to the Message Send-Receive task through the Message Send-Receive Input Buffer (see Section 4.2.2.1). In the event of a hardware error condition being detected while a received data packet is being processed, status variables reflecting the health and availability of layers and ICISes are updated immediately within the ICIS_RM task. An error detection report is passed to another task responsible for additional fault isolation functions and for managing further responses to the fault which can be deferred until remaining received packets are processed.

ICIS RM functions which are less time critical have been implemented as part of this second, lower priority task referred to as the ICIS_LOCAL_MANAGER. Included in this task are those functions required to manage responses to detected error conditions and those required to re-initialize the ICIS hardware. This task maintains an interface with the Network Manager task wherein layer faults can be reported to the Network Manager and updates to the status of communication layers can be passed to the ICIS_LOCAL_MANAGER which will, in turn, update local status variables. This task also maintains an interface such that the system software managing the redundancy of the

core FTP can make requests to have a particular ICIS "re-aligned" following the recovery of a channel. The ICIS_LOCAL_MANAGER task, in response to hardware errors detected by the ICIS_RM task, will also perform the less time critical functions such as error logging and scheduling of self tests to either further isolate the fault and/or to determine when a fault is no longer present. The execution of these self-tests and the evaluation of their results are included in the set of ICIS_LOCAL_MANAGER functions. Finally, this task is responsible for re-initializing an ICIS when the results of its self-tests indicate that a fault has been removed or subsided.

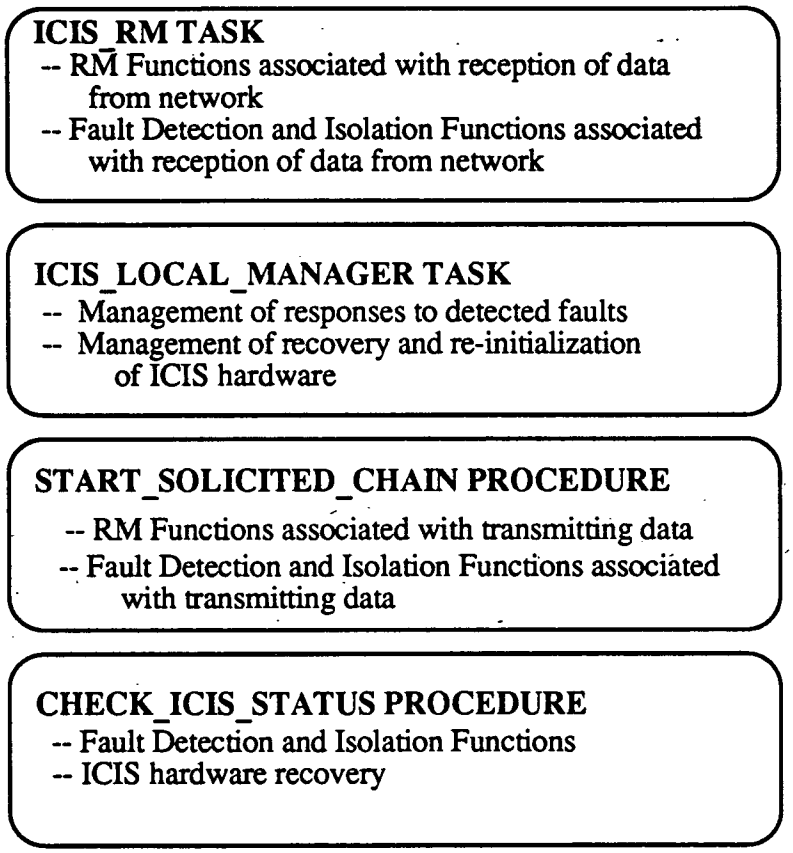


Figure 4-12. ICIS Redundancy Management: Mapping of Functional Requirements to Software Objects

Two other ICIS RM functions are made available in the form of simple subroutines which execute in the context of the task calling them. Any task which executes an ICIS solicited chain of instructions (e.g., to transmit data packets or to perform network management functions involving the network nodes) must call the START_SOLICITED_CHAIN procedure. This procedure manages the redundancy of the communication hardware during the data transmission process and it also provides FDI functions related to arbitration for network possession and the transmission of data on the network. A second subroutine, CHECK_ICIS_STATUS, is available to be called by any task which detects a problem in executing solicited chains or in accessing the ICIS memory.

CHECK_ICIS_STATUS provides FDI functions and, to a lesser extent, it provides an ICIS re-initialization function in the case of a detected possession default state.

4.2.3.1 ICIS_RM TASK

Input:

Redundant data packets received from IC network
Redundant status from ICIS hardware

Output:

Source Congruent representation of received data
Fault detection and isolation information for IC hardware

Description:

This task provides the redundancy management functions associated with the reception of data from the IC network. The IC source congruency function applied to the received data and the FDI function related to the reception of network data have been combined within the body of this task. This software directly interfaces with the ICIS hardware to get redundant copies of data and status and transforms this input into a source congruent representation and provides error detection and fault isolation information as output. The task executes only on the IOP. The structure of this task is shown in Figure 4-13.

Each iteration of the ICIS_RM task's main loop begins with a check to see if new data packets have been received since the last time the task ran. Before making the check, however, a determination must be made as to which channel's ICISes are to be included in this process of polling for new input. The data from a particular channel's ICIS is considered usable if (1) the channel is considered non-faulty by the core FTP redundancy management services, (2) the ICIS is considered non-faulty on the basis of status variables maintained by the ICIS RM software, and (3) the ICIS has been initialized. The primary means of detecting newly received packets is a comparison of two variables - Last_Pack and First_Pack. The First_Pack variable maintains an index value for the last packet processed by the ICIS_RM task. The Last_Pack variable maintains an index for the last packet received by the ICIS and is updated by the ICIS instruction sequencer as part of the execution of the unsolicited chain of instructions responsible for accepting packets from the IC network. Both indices take on values 1..N, where N is the number of slots in the circular buffer allocated within the ICIS dual-port memory and used to buffer received packets of data. The First_Pack variable is updated by the processor during the course of processing the received packets and should always be channel congruent. However, since the Last_Pack variable is maintained in ICIS dual-port memory and is updated by the ICIS instruction sequencer, the redundant copies of this variable can not be assumed to be congruent. The comparison of First_Pack is made against a single-sourced exchanged copy of the Last_Pack variable. A separate comparison is made for each channel

determined to have usable ICIS data. If any comparison indicates Last_Pack and First_Pack are not the same, then it is assumed that new input needs to be processed.

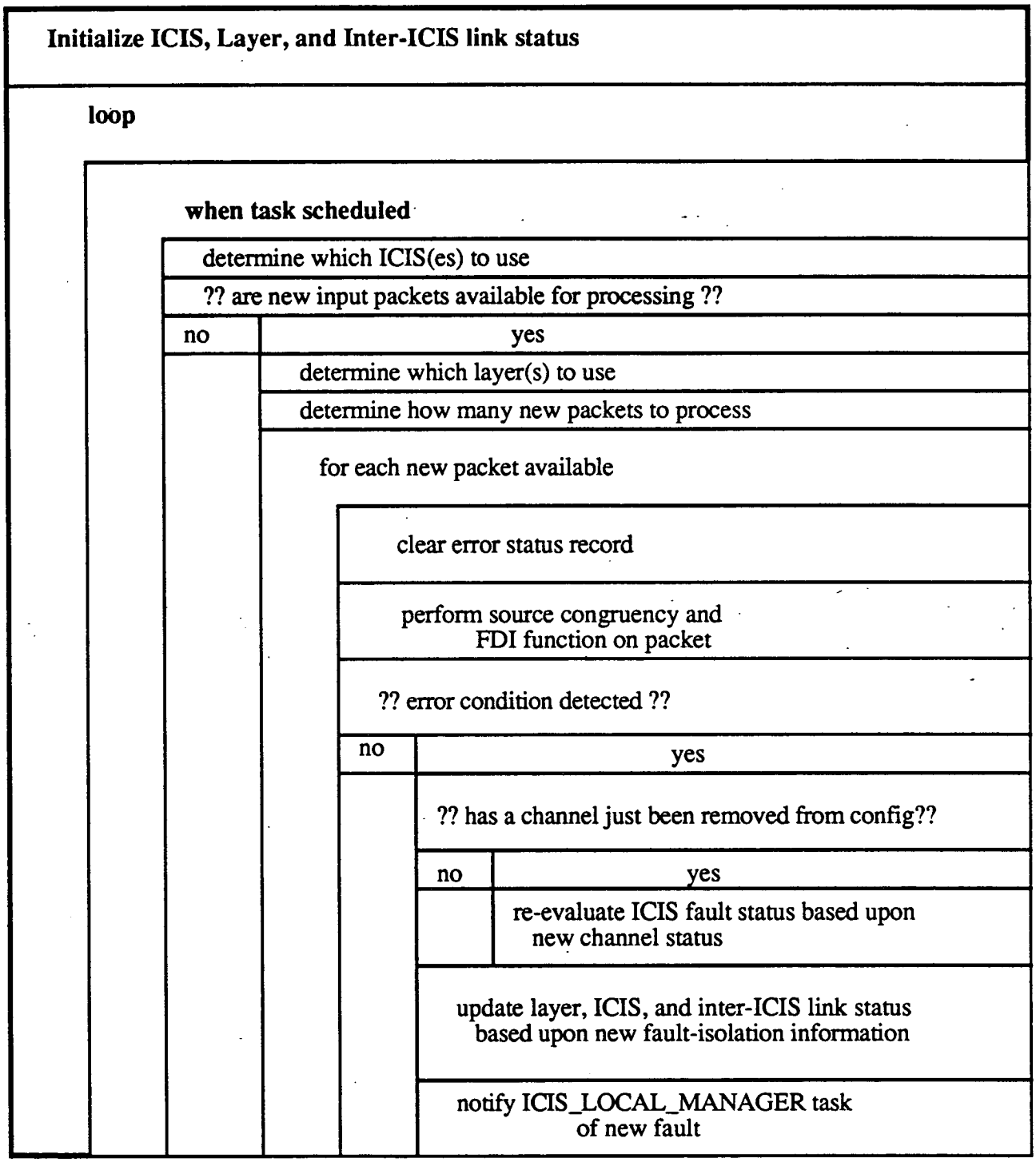


Figure 4-13. Structure of ICIS_RM Task Body

(NOTE: A registered bit of information, the Unsolicited_Input_Received (UIR) bit, is available on the ICIS and is set when the ICIS receives an input packet while operating in the unsolicited mode. This bit can be used to check for new data. However, this mechanism was not used to determine when input packets need to be processed, due to an anomaly in the ICIS hardware implementation. The ICIS hardware will advance through an INPUT instruction in the Unsolicited Chain due to mode context switches - Unsolicited to Solicited and back to Unsolicited modes - independently of the reception of a network packet. This advancement of INPUT instructions results in the "skipping" of the input buffer associated with the "skipped" INPUT instruction. This input buffer obviously has no data to process, yet it must at least be processed to the point of determining that it is a "null" input packet. The UIR bit does not get set due to this "skipping" of input instructions and if it were used alone for polling purposes, multiple null packets could accumulate before the ICIS_RM task decided to process these packets. The Last_Pack variable, on the other hand, does get updated when INPUT instructions are skipped.)

If the check for newly received packets indicates there is currently no processing to be performed, the iteration of the ICIS_RM task is completed and the task is suspended. Otherwise, the task will continue to execute until all of the newly received packets have been processed. The number of packets to be processed is a function of the current values of First_Pack and Last_Pack. Again, care must be taken to ensure that the channels of the FTP use a congruent value of Last_Pack which may be non-congruent either because there is an ICIS failure or because the instruction sequencers for the different channels' ICISes are not synchronized and therefore don't update the Last_Pack variable at exactly the same time. In the case where two or three ICISes are being used, the Last_Pack variable is read repeatedly with an implicit voted exchange until two successive reads return the same value. When only one ICIS is being used, the Last_Pack is read once using an implicit single channel select.

An inner loop of the ICIS_RM task's main loop is executed where each iteration of this inner loop provides the necessary processing for performing the source congruency and FDI functions on an individual input packet. The details of the Process_Packet process as applied to a single input packet are provided in Section 4.2.3.1.1. An error record is maintained for the processing of an individual packet and accumulates information derived by the FDI function. The processing of the individual packet nominally culminates in the writing of a source congruent copy of the received data into a buffer in local processor memory which provides an interface to the Message Send-Receive task (refer to Section 4.2.2.1). (There are certain error conditions where no data can be derived, and the processing of "null" data packets discussed above does not result in the updating of the Message Send-Receive input buffer.)

If an error condition is detected during the Process_Packet procedure, a check is made to determine if a fault isolated to an ICIS is not really the result of a channel failure. As documented below, much of the FDI processing of ICIS data and status relies on the use of

the data exchange voter hardware and accompanying error registers to detect non-congruencies across layers and ICISes. Attributing the cause of a non-congruency indication to a fault in an ICIS (i.e., there really is non-congruent data or status) can be confounded by the fact that a channel failed during the time that the data or status was being processed. This check consists of calls to the FTP redundancy management services to confirm the synchronous presence of a channel and consists of an actual voted exchange test and abbreviated data exchange error latch analysis to confirm that a voted exchange of congruent data does not set bits in the error latches. A recording of a fault isolated to an ICIS which is found to be confounded by a channel failure is purged. No fault reports are logged nor are any responses made (i.e., updates to status variables to mask out data from the ICIS, or scheduling of self-tests).

If a new error condition is detected during the processing of a packet, responses are made immediately within the packet processing inner loop of the ICIS_RM task. The status variables indicating the fault status need to be updated immediately as their values will determine which set of redundant copies of data and status will be eligible for use in processing subsequent input packets. The Healthy_Layer or the Healthy_ICIS status variables are updated when a layer or a total ICIS, respectively, is determined to have failed. The response to a fault isolated to the region of a partial ICIS or one of the inter-ICIS links (e.g., the ICIS seems to receive fault-free data on two out of three layers and other ICISes receive fault free data on all three layers) is more complex. If there is only one ICIS available and a partial ICIS failure is detected, then the data from the layer involved in the fault will be marked "unavailable" for subsequent packet processing. Note that the fault response in this case does not include any attempt to implicate the layer as faulty as there is not enough information to do so. Another case in which the layer data is marked "unavailable" is where two out of three ICISes receive faulty data from the same layer while the non-faulty copy is in complete agreement with redundant copies received on at least one other layer. Again, the network layer can not be presumed to be failed on the basis of this evidence; more likely there is a problem in the inter-ICIS data links associated with that layer. In other cases involving faults isolated to a partial ICIS region where there is at least one other fault-free ICIS available to the FTP, all data from the ICIS with the partial failure will be made "unavailable" for processing subsequent input packets. No response is made when it is determined that a double simultaneous fault has occurred or when the FTP sourcing the received packet sent confusing and non-congruent data across two layers.

If any error condition is detected while processing an individual input packet, an error report including the fault isolation information is passed along to the ICIS_Local_Manager (ILM) task. This inter-task communication is implemented using a circular buffer structure for passing the error reports and the setting of an event to trigger the execution of the ILM task once the higher priority ICIS_RM task has finished its current execution wherein all

currently received input packets are processed. The subsequent actions taken in response to the detected error condition are managed by the ILM task and are documented in Section 4.2.3.2.

When all the outstanding input packets have been processed, the First_Pack variable is updated with the index of the last processed packet and the outer loop of the ICIS_RM task comes back to the point where the task suspends itself.

4.2.3.1.1 Process: Packet_Process

Input:

Redundant copies of individual input packets from IC network

Output:

Source congruent copy of input packet
Fault detection and isolation information

Description:

This process embodies the redundancy management functions related to the processing of redundant data and status information associated with a single, received IC network packet. The structure of this process is shown in Figure 4-14. The process is invoked from the main loop of the ICIS_RM task for each new element of the input packet buffer determined to have been "updated" by the ICIS instruction sequencer either in response to actual data received from the network or to the mode switching anomaly previously described. This process is responsible for:

1. determining upon which layers data was received,
2. analyzing the redundant byte count values associated with the redundant packets,
3. analyzing the redundant SDLC protocol status associated with the redundant packets,
4. analyzing the redundant data values associated with the redundant packets.

Each of these subprocesses is implemented as a separate subroutine which is only called by the Packet_Process routine. It should be noted that subprocesses are invoked only conditionally on the basis of the outcome of preceding subprocesses. For example, if the subprocess responsible for determining upon which layer(s) data was received indicates that data was not received on any of the three layers (i.e., this is a "null" packet resulting for a ICIS unsolicited-to-solicited mode transition), then there is no need to perform any of the other subprocesses. Also, the parameters used by a subprocess may be determined on the basis of the outcomes of previous subprocesses. For example, if a layer is determined to be faulty during the subprocess involving the analysis of byte count values, then the data from that layer will not be used within the subprocess which does the voting/selecting of the data values in forming a single, channel-congruent representation of the received data.

The output of this process consists of an error status record identifying any error conditions detected while processing the current input packet, and a single channel-congruent representation of the received data which is conditionally placed in the Message Send-Receive Input Buffer. The error status record is a function of the error detection results of the byte count, SDLC protocol, and received data analyses and their inter-correlations. Only the subprocess responsible for forming the congruent data representation updates the Message Send-Receive Input Buffer.

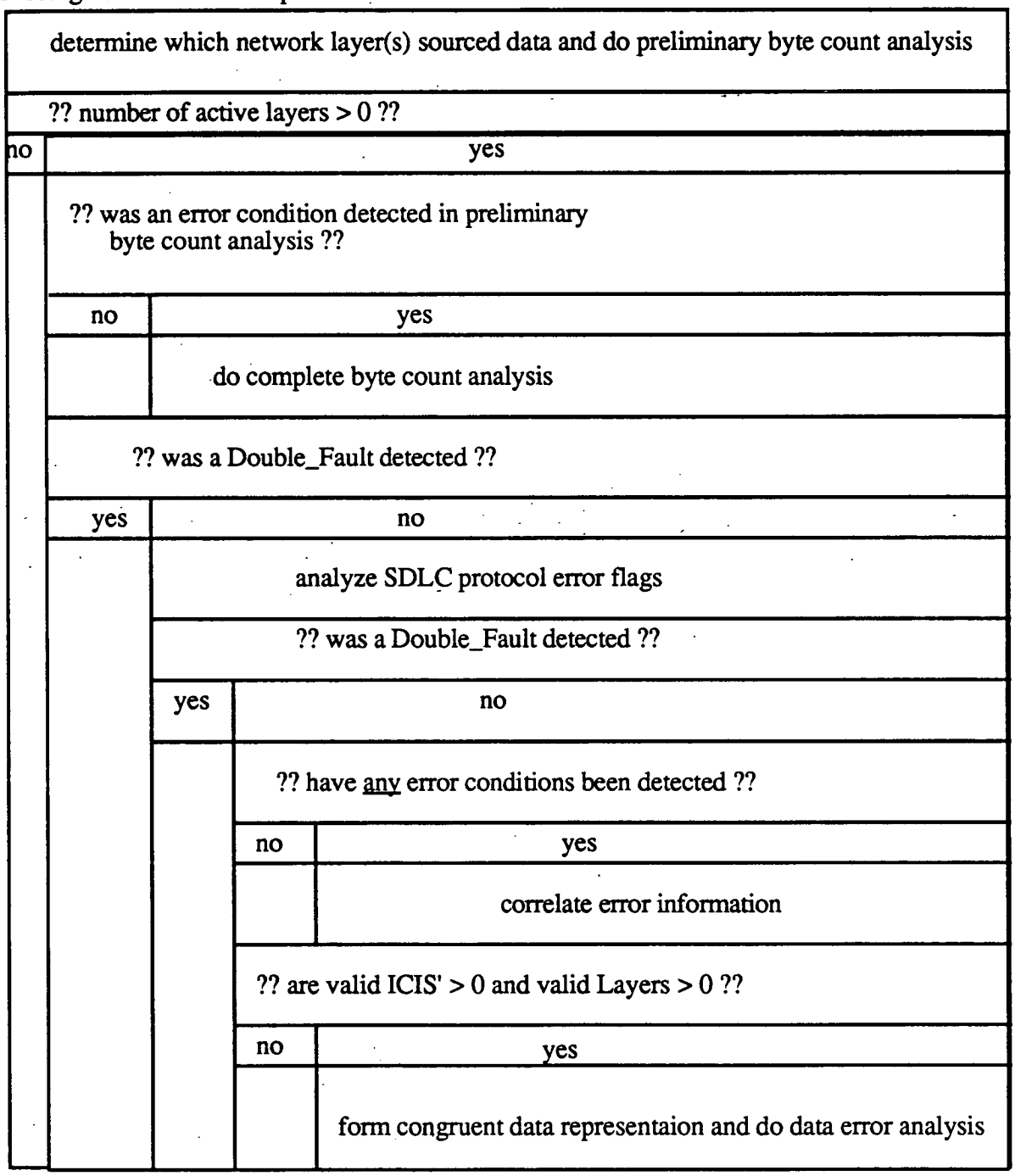


Figure 4-14. Structure of Packet_Process

4.2.3.1.2 Process: Get_Active_Layers

Input:

Redundant copies of individual input packets from IC network
Indication of which network layers are non-faulty
Indication of which channels' ICISes are available.

Output:

Indication of which network layers sourced data for this packet
Fault status from comparisons of redundant byte counts
Congruent byte count value for received packet (may be modified in later processing of faulty packets)

Description:

This process is responsible for determining which, if any, layers sourced data for the current input packet being processed. The need for this process arises from the fact that the transmitting sites in the AIPS distributed system can be of mixed redundancy. A simplex site will transmit data only on one layer whereas a triplex site will transmit on all three network layers. The redundancy level of a transmitting site is not known *a priori* and must be analytically determined on the basis of the redundant data received from the network for each and every input packet. In addition, the possible "null" packet scenario resulting from the unsolicited-to-solicited mode transition of the ICIS must be detected by the Get_Active_Layer process. The process, in this "null" packet case, returns an indication that zero layers have active data, resulting in the termination of any further processing of the current input packet.

The determination as to whether a network layer sourced data for a particular input packet is based upon an analysis of the byte count value associated with each copy of the received input packet. While receiving each redundant data packet from the separate network layers, the ICIS hardware maintains a count of the number of bytes of information associated with each incoming SDLC packet. At the completion of an ICIS INPUT instruction, the independent byte count values associated with the separate layer interfaces are written into the ICIS dual-port memory as part of the recorded input packets' headers. When an INPUT instruction is completed and no SDLC packet data was received for a particular layer, then the byte count value recorded is 4 - representing only the number of bytes in an ICIS-generated header for a received packet. (This header includes the byte count value, status values associated with the hardware device performing the SDLC protocol, the ICIS Chain_Status_Register value, and a time tag value.)

For each fault-free network layer, the redundant byte counts associated with valid (i.e., fault-free channels and ICIS hardware) ICISes are voted/selected. The "LMN space voter/select" hardware built into the AIPS hardware is used for this operation. A particular

layer is considered to be "inactive" (i.e., did not receive input for this current input packet) only if the voted/selected byte count for the layer is a value of either 0 or 4 by consensus. That is, all valid ICISes must have seen no data sourced on the layer before it is marked inactive. The voted byte count for a layer may be 0 or 4, but if a voter error indicates an incongruous value across the channels then the layer is still considered active so that its data will be included in the more elaborate fault detection and isolation processing of the byte count, SDLC protocol, and data values which follows.

Note that this same processing of the redundant byte count values is needed to derive a single, channel-congruent byte count value used in processing the data within the input packets. Thus, instead of doing the same vote/select operations on the byte count values for both determining which layers are active and to determine "the real" number of data bytes received, both pieces of information are gathered at the same time within this process if possible. The byte count values voted/selected across the channels for each layer are compared against one another. Under no-fault conditions, all redundant byte counts will be equivalent and the congruent byte count value is returned for use in all subsequent processing of the input packet. The detection of any byte count inequalities or byte count range check failures results in the return of a error detection flag that will be used to invoke a more complete analysis of the byte counts to localize the fault.

4.2.3.1.3 Process: Byte_Count_Analysis

Input:

Redundant copies of individual input packets from IC network
Indication of which network layers have valid data
Indication of which channels' ICISes are available

Output:

Channel-congruent representation of byte count for input packet
Byte Count Fault Detection and Isolation record

Description:

This is a subprocess called by the Packet_Process process only when the preliminary analysis of the redundant byte count values associated with an input packet indicates that there is a fault to be more completely analyzed and that the "real" number of bytes received, as determined in the preliminary analysis, may need to be re-evaluated on the basis of the results of this fault isolation operation. A byte count is considered erroneous and indicative of a fault if it deviates from the majority byte count value (i.e., the byte count value observed most frequently in the set of redundant input packets from the fault-free layers and ICISes). This process detects erroneous byte counts and locates them in the two-dimensional space formed by the crossing of the available network layers and the available redundant ICISes of the receiving site. The number of "minority" byte count values and

their location in this "Layer x ICIS" matrix are used for isolating the fault to a fault isolation region. This process is also responsible for detecting the case in which a double simultaneous fault is present and the case where there is no single majority byte count but two byte count values are "TIED" in terms of frequency of occurrence.

The possible byte count error states returned by this process are:

NONE -- no byte count errors

DOUBLE_FAULT -- indicative of double simultaneous error condition; no further processing to be applied to this input packet

POINT -- one minority byte count found, or 2 out of 3 redundant byte counts for a particular layer are minorities

TOTAL_LAYER -- multiple minority byte counts in one layer; the layer will not be used

TOTAL_ICIS -- unanimous, multiple minority byte counts in one channel's ICIS; the data from this ICIS will not be used

TIE_LAYER -- all byte counts within a layer are the same yet different from the byte count of a second layer which also has consistent byte counts; to be resolved with SDLC error analysis

TIE_ICIS -- all byte counts within a channel's ICIS are the same yet different from the byte count of a second channel's ICIS which also has consistent byte counts; to be resolved with SDLC error analysis

This information will be later coordinated with the fault status associated with the SDLC protocol analysis and the analysis of the actual packet data before a conclusive determination of a fault location is made.

The byte count analysis algorithm is summarized in Figure 4-15 and discussed in detail below.

1. The "Layer x ICIS" matrix of byte count values is formed by reading the individual byte count values stored in the header of each redundant copy of the input packet being processed. This data is read in a "single channel select" mode from only those layers and ICISes currently considered to be providing valid input packets. As the byte count values are read, a tabulation of the frequency of occurrence of each distinct byte count value is maintained.

FREQUENCY OF MINORITY BYTE COUNTS	NO TIE	TIE
0	NO FAULT	N / A
1	POINT FAILURE FOR SINGLE LAYER, ICIS LOGGED.	IF NUMBER OF LAYERS = 2 THEN TIE_LAYER ELSIF NUMBER OF ICIS' = 2 THEN TIE_ICIS
>1	IF ALL MINORITIES IN 1 LAYER THEN TOTAL_LAYER FAILURE ELSIF ALL MINORITIES IN 1 ICIS THEN TOTAL_ICIS FAILURE ELSE DOUBLE FAULT	IF ALL BYTE COUNTS EQUIVALENT WITHIN EACH ACTIVE LAYER THEN TIE_LAYER ELSIF ALL BYTE COUNTS EQUIVALENT WITHIN EACH ACTIVE ICIS THEN TIE_ICIS

Figure 4-15. Algorithm for Byte Count Analysis

2. The frequency of occurrence data is scanned to determine the majority byte count value and to detect a TIE byte count situation.

3. A preliminary check of the frequency of occurrence of the majority byte count is made to detect a double-fault case. The threshold for the frequency of occurrence is a function of the number of current valid layers and ICISes. For example, if there are currently three valid layers and three valid channels (i.e, a triplex is receiving a transmission from another triplex and no previously detected IC error conditions have been noted), then at least six out of the nine redundant input packets must have the same byte count or there is a double simultaneous fault condition present in the system.

4. If there is no TIE condition, then the byte count status to be returned is formed according to the location of any minority byte count values in the "Layer x ICIS" matrix. Obviously, if there are no minority byte count values then a byte count status of NONE is returned. If there is only one minority byte count value, then a POINT byte count status is returned along with the identification of which ICIS and layer are involved. The determination of the returned status is more complex when there are multiple minority byte count values involved. If all the dissenting byte counts are located in the same layer then either a TOTAL_LAYER status is returned when all redundant byte counts for packets received on that layer are minorities, or a POINT status is returned in the case where only two out of three of the byte counts for the layer are minorities. If all the dissenting byte counts are located in the same ICIS, then a

TOTAL_ICIS status is returned. If all minorities are not located either all in one layer or all in one ICIS, then a DOUBLE_FAULT status return is made.

5. If a TIE condition is indicated, a determination must be made as to whether the TIE is between two sets of byte counts distributed evenly between two layers (TIE_LAYER), or between two ICISes (TIE_ICIS). Any other distribution of byte count values is considered indicative of a DOUBLE_FAULT condition. In the case of a TIE, then both byte count values are range checked to determine if they are legal byte counts. If one value is determined to be illegal and the other is legal, the legal value is considered to be the "real" byte count to use.

4.2.3.1.4 Process: SDLC_Error_Analysis

Input:

Redundant copies of individual input packets from IC network
Indication of which network layers have valid data
Indication of which channels' ICISes are available

Output:

SDLC protocol error detection and isolation record

Description:

This subprocess is responsible for analyzing any SDLC protocol error indications associated with the input packet currently being processed. For each redundant copy of the input packet, a single bit of information is stored in the header of the input packet which indicates whether or not the hardware device implementing the SDLC protocol detected an error while receiving the packet. These protocol errors consist of:

1. Cyclical Redundancy Checks (CRC) - The transmitting SDLC device forms a CRC value for the outgoing packet and appends the value to the end of the packet. The receiving SDLC device performs the same CRC operation on the incoming packet data. If the determined CRC value does not match the value appended to the packet, then there was a transmission error.

2. Data overrun - The receiving SDLC device was not read quickly enough by the controlling ICIS hardware to keep up with the incoming SDLC serial bit stream.

3. Format error - The incoming bit stream does not conform to the SDLC protocol (e.g., no closing flag detected after the maximum possible number of data bytes per packet was received).

These three error conditions are ORed together into one status bit (RCVR_ERR) in the SDLC device's Interface_Register (IR), a copy of which is stored in the input packet's header by the ICIS hardware at the conclusion of an INPUT instruction.

The analysis process consists of first reading all redundant copies of the RCVR_ERR bit from the input packets associated with the currently valid layers and ICISes. These reads are performed with the "select single channel" mode (i.e., an implicit single-sourced exchange is performed). The number of packets with protocol error indications is noted as well as the location of the faulty packet(s) in the "Layer x ICIS" matrix. The SDLC error status returned is a function of the number and location of packets with protocol errors:

NONE -- no SDLC errors detected

DOUBLE -- multiple SDLC errors detected and they aren't either all in one layer or all in one ICIS; further processing for this packet is cancelled

POINT -- single SDLC error detected or 2 out of 3 packets for a particular layer have errors; the identification of the layer and ICIS (or ICISes) involved is also returned

TOTAL_LAYER -- multiple SDLC errors all found within one layer

TOTAL_ICIS -- multiple SDLC errors all found within one channel's ICIS

The SDLC analysis algorithm is summarized in Figure 4-16.

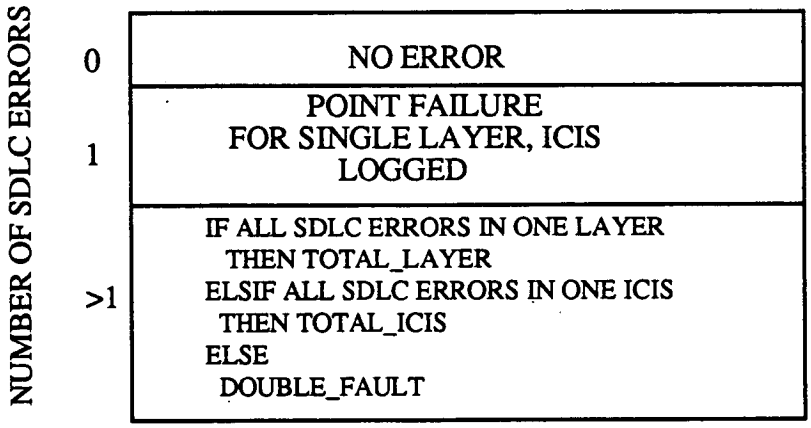


Figure 4-16. Algorithm for SDLC Analysis

4.2.3.1.5 Process: Correlate_Error_Information

Input:

Error status from Byte_Count_Analysis
Error status from SDLC_Error_Analysis

Output:

Error detection and fault location record
Indication of from which layers and ICISes to process data
Final byte count value (following resolution of TIE byte counts)

Description:

This subprocess is called from within the Packet_Process code only if an error detection indicator has been set during either the processing of the byte count values or the SDLC error flags. The subprocess is responsible for correlating the fault location information returned from the byte count and SDLC error processing for the current input packet. The execution of this subprocess provides an updated fault location record which is based upon the composite information provided by the byte count and SDLC error information. It also provides an updated indication of which layers and ICISes are to be used in forming the single, congruent representation of the redundant data. An additional operation provided by this subprocess is the conditional resolution of the ambiguous TIE byte count situation where the inclusion of the SDLC error information may help in determining which set of packets with one particular byte count value might be faulty.

The following two tables (Figures 4-17 and 4-18) list the actions taken depending upon the byte count error status and the SDLC error status. Figure 4-17 indicates the logic used in deactivating channels or layers based upon these error status values. Note that the deactivation of a channel or layer refers to the change in status regarding the valid channels and layers to be used in forming congruent data. That is, if a channel or layer is deactivated, its data is not used in the subsequent vote/comparison process. Figure 4-18 lists the rules used to determine a collaborative error status based upon both the byte count status and the SDLC status.

In attempting to understand the derivation of the actions specified in these two tables, consider two transparent sheets of plastic each of which has a 3 x 3 matrix inscribed to represent the "Layer x ICIS" matrix. One of the sheets will represent the fault status returned by the byte count analysis and the other indicates the fault status returned by the SDLC error analysis. A fault is indicated by darkening the appropriate part of the matrix associated with the fault's location:

-- a point fault has only one or two elements of the matrix darkened (all in same column)

-- a total ICIS fault has a complete column darkened

-- a total layer fault has a complete row darkened

The two transparencies are overlaid forming a single matrix. The indicated faults (i.e., darkened areas) of the composite are localized with the same basic rules used in the original

SDLC ERROR STATUS

BYTE COUNT ERROR STATUS	NONE	POINT	TOTAL ICIS	TOTAL LAYER
NONE	NOTHING	DEACTIVATE EITHER LAYER OR ICIS DEPENDING UPON TOTAL RESOURCES AVAILABLE	DEACTIVATE THE ICIS	DEACTIVATE THE LAYER
POINT	DEACTIVATE EITHER LAYER OR ICIS DEPENDING UPON TOTAL RESOURCES AVAILABLE	IF BOTH POINTS SAME THEN DEACTIVATE EITHER LAYER OR ICIS DEPENDING UPON TOTAL RESOURCES AVAILABLE ELSE TREAT AS TOTAL ICIS, TOTAL LAYER, OR DOUBLE FAULT BASED UPON LOC. OF POINTS	IF SAME ICIS FOR BOTH ERRORS THEN DEACT. ICIS ELSE DOUBLE FAULT	IF SAME LAYER FOR BOTH ERRORS THEN DEACT. LAYER ELSE DOUBLE FAULT
TOTAL LAYER	DEACTIVATE THE LAYER	IF SAME LAYER FOR BOTH ERRORS THEN DEACT. LAYER ELSE DOUBLE FAULT	DOUBLE FAULT	IF SAME LAYER FOR BOTH ERRORS THEN DEACT. LAYER ELSE DOUBLE FAULT
TOTAL ICIS	DEACTIVATE THE ICIS	IF SAME ICIS FOR BOTH ERRORS THEN DEACT. ICIS ELSE DOUBLE FAULT		DOUBLE FAULT
TIE ICIS	DOUBLE FAULT	DEACTIVATE ICIS WITH SDLC ERROR, USE BYTE COUNT OF NONERRONEOUS CHANNEL		DOUBLE FAULT
TIE LAYER	SOURCE GPC FAULT NO USABLE DATA	DEACTIVATE LAYER WITH SDLC ERRORS, USE BYTE COUNT OF NONERRONEOUS LAYER	DOUBLE FAULT	DEACTIVATE LAYER WITH SDLC ERRORS, USE BYTE COUNT OF NONERRONEOUS LAYER

Figure 4-17. Layer/Channel Deactivation

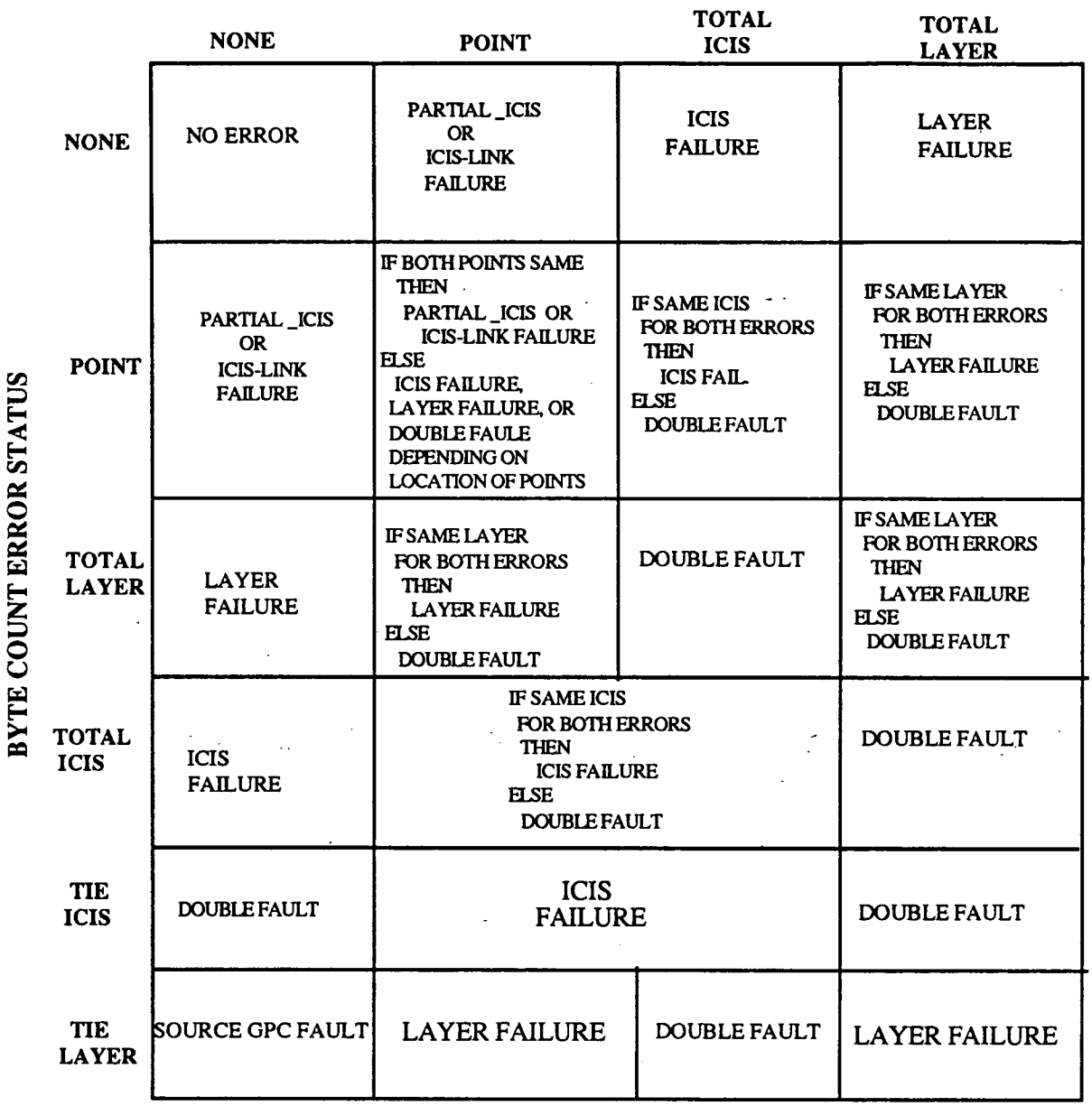


Figure 4-18. Error Status Determination

byte count and SDLC error analysis. All errors must be localized either to one and only one row or column (i.e., one layer or one ICIS); else a double fault condition is indicated.

The handling of the POINT-type fault indicators is relatively more complex than the other types. A POINT fault is indicative of either a partial ICIS failure or a failure in the inter-ICIS communication links used to distribute incoming network data to the redundant ICISes of an FTP. Thus, a POINT-type failure may involve more than one channel's ICIS in the cases where one of the inter-ICIS links is faulty within the daisy-chain before two ICISes receive the data, but at least one ICIS receives fault-free data on that particular layer. When two ICISes are involved and there are at least two fault-free network layers available, the layer is deactivated although the layer is not considered faulty (i.e., there is nothing the Network Manager could do to alleviate this problem). When two ICISes are involved and there is only one layer available, then two ICISes are deactivated and the site is left to receive only one copy of the input packet. If only one channel's ICIS is implicated in the fault, then whether the layer or the ICIS is deactivated must be decided. When there is more than one ICIS available the ICIS is deactivated; otherwise the layer is deactivated.

4.2.3.1.6 Process: Get_Congruent_Data

Input:

Redundant copies of individual input packets from IC network
Indication of which fault-free layers sourced data
Indication of which channels' ICISes are available
Byte counts for this input packet

Output:

Error detection and fault location record
Single, congruent representation of received packet data

Description:

This subprocess attempts to produce a single, channel-congruent representation of the data included in a received packet. When miscomparisons in the redundant copies of data are detected, a detailed analysis of the miscomparisons is invoked for fault isolation purposes. An additional function provided by this subprocess is the checking of the actually observed network layer activity against the expected layer activity as determined by the layer redundancy encoding placed in the data packet by the transmitting FTP. This check is done to provide a mechanism to determine when data failed to show up on a particular layer when it should have (i.e., there is a layer fault). The check is made only after a congruent representation of the layer redundancy encoding is formed.

The AIPS hardware support for performing vote/select operations across redundant layers within a single ICIS and for performing vote/select operations across redundant ICISes is used extensively in this subprocess. Only data from layers and ICISes considered to be fault-free is included in the vote/selects. For example, when a triplex site receives a transmission from a simplex site on one layer, the data is selected only from that single layer within each ICIS and then voted across the three channels of the receiving FTP. In the case where a triplex transmits to another triplex the data is first voted across all three layers within each ICIS and the resulting voted values are again voted across the channels. (Refer to Figure 4-19, which indicates the types of votes/selects performed as a function of the number of valid layers and valid ICISes involved in the current input packet.) Note that both the vote and the select operations across channels imply that a data exchange function is executed and the resulting data is guaranteed to be congruent across the channels. The vote/selects are done implicitly during the movement of the data from the ICIS dual-port memory into the local processor memory. The processed packet is stored as one element in the Message Send-Receive Input Buffer (refer to Section 4.2.2.1).

There are error latches associated with both the voters used to operate on data across layers and the voters used to operate on data across channels. These latches are used to accumulate status indicative of miscomparisons in the data moved out of the ICIS dual-port memory. The error analysis logic is summarized in Figure 4-20. The results of this error analysis are combined with the error detection and fault isolation information already accumulated during the processing of this packet (i.e., from byte count and SDLC error analyses) to determine a final error status and fault location record for this processed packet.

Once the data has been made source congruent, the layer redundancy encoding is extracted and compared against the observed layer activity associated with this packet reception. If a layer was expected to source data for this packet according to the layer redundancy encoding but no data was detected on this layer, then the layer is flagged as faulty. This information is integrated into the final error status record.

Only if the final error status accumulated after all of this processing indicates that there is no double fault condition and no unresolvable ambiguity in the received data, is the new input data passed to the Message_Send_Receive task by updating the FDIR_index in the Message Send-Receive Input Buffer. An event is set to signal the Message_Send_Receive task that new input is available for processing. If no valid data can be passed, the buffer index is not updated and the Message_Send_Receive task is not notified.

NUMBER OF VALID CHANNELS

NUMBER OF VALID LAYERS

	1	2	3

1	EXCHANGE VALID CHANNEL'S DATA FROM SINGLE DATA PACKET.
	(IF NO PREVIOUS ERROR
	THIS IS SIMPLEX - SIMPLEX)
		COMPARE DATA INTERCHANNEL.
		EXCHANGE E_L'S.
		IF MISCOMPARE THEN
		NO USABLE DATA
		IF NO PREVIOUS FAULT
		THEN
		PARTIAL_ICIS OR ICIS_LINK FAILURE
		ELSE
		DOUBLE FAULT
			VOTE DATA INTERCHANNEL.
			EXCHANGE E_L'S.
			IF MISCOMPARE THEN
			IF NO PREVIOUS FAULT
			THEN
			PARTIAL_ICIS OR ICIS_LINK FAILURE
			ELSE
			DOUBLE FAULT

2	COMPARE DATA ACROSS LAYERS INTRACHANNEL.
	EXAMINE E_L.
	IF MISCOMPARE THEN
	NO USABLE DATA
	IF NO PREVIOUS ERROR
	THEN
	SOURCE_GPC FAILURE
	ELSE
	DOUBLE FAULT
		COMPARE DATA ACROSS LAYERS INTRACHANNEL AND THEN COMPARE RESULT ACROSS CHANNELS.
		EXCHANGE E_L'S FOR LMN INTRACHANNEL COMPARISONS.
		IF MISCOMPARE THEN
		IF NO PREVIOUS ERROR
		IF ONE CHANNEL MISCOMPARE
		PARTIAL ICIS OR ICIS-LINK FAIL.
		ELSIF BOTH CHANNELS MISCOMPARE
		SOURCE_GPC FAIL
		NO USABLE DATA
		END IF
		ELSE
		DOUBLE FAULT
		END IF
		END IF
		
		IF MISCOMPARE BETWEEN CHANNELS
		IF NO INTRACHAN MISCOMP.
		DOUBLE FAULT
		ELSIF ONLY ONE CHANNEL HAS INTRACH MISCOMP
		THEN
		EXCHANGE DATA FROM CHAN. WITH 2 CONGRUENT LAYERS
		END IF
		END IF
			COMPARE DATA ACROSS LAYERS INTRACHANNEL AND THEN VOTE ACROSS CHANNELS.
			EXCHANGE E_L'S FOR LMN INRACHANNEL COMPARISON..
			COUNT # OF CHANNELS WITH MISCOMPARES.
			CASE COUNT IS
			WHEN 0, CHECK FOR ERROR IN VOTE ACROSS CHANS.
			IF ONE CHANNEL MISCOMP THEN ICIS FAIL
			IF >1 CHAN. MISCOMP. THEN DOUBLE FAULT
			WHEN 1, IF NO PREVIOUS ERRORS THEN
			PARIAL_ICIS OR ICIS LINK FAILURE
			ELSE
			DOUBLE FAULT
			WHEN 2, DOUBLE FAULT
			WHEN 3, IF NO MISCOMP. ACROSS CHANNELS
			THEN
			SOURCE_GPC FAIL
			NO USABLE DATA
			ELSE
			DOUBLE FAULT

3	VOTE DATA INTRACHANNEL ACROSS LAYERS. EXCHANGE RESULT FROM VALID CHAN.
	EXAMINE LMN INTRA E_L.
	IF ONE LAYER MISCOMP.
	THEN
	IF NO PREVIOUS ERROR
	PARTIAL ICIS OR ICIS LINK FAILURE
	ELSE
	DOUBLE FAULT
	END IF
	ELSIF >1 LAYER MISCOMP.
	THEN
	DOUBLE FAULT
		VOTE DATA ACROSS LAYERS INTRACHANNEL AND VOTE RESULT ACROSS CHANNELS.
		EXCHANGE LMN INTRA ERROR LATCHES.
		IF THERE ARE ANY MISCOMPARES INDICATED BY THESE ERROR LATCHES:
		-COUNT THE # OF LAYERS WITH MISCOMPS.
		-COUNT THE # OF CHANNELS WITH MISCOMPS
		-USE THE LOGIC ILLUSTRATED IN FOLLOWING TABLE TO ISOLATE ERRORS AND TO DETERMINE WHETHER DATA IS USABLE.
		IF THERE ARE NO INTRACHANNEL MISCOMPARES YET THERE IS A MISCOMPARISON ACROSS CHANNELS THEN
		IF ONLY ONE CHANNEL MISCOMPARES ICIS FAIL
		ELSE DOUBLE FAULT	

Figure 4-19. Votes/Selects Used for Layer/Channel Combinations

NUMBER OF CHANNELS WITH MISCOMPARES ACROSS LAYERS

NUMBER OF LAYERS WITH MISCOMPARES	1	2 OUT OF 3	ALL
1	IF NO INTERCHANNEL ERRORS THEN PARTIAL_ICIS OR INTER-ICIS LINK FAILURE ELSIF BOTH INTRA- AND INTER- CHANNEL ERRORS POINT TO SAME ICIS THEN ICIS FAIL ELSE DOUBLE FAULT	IF NO INTERCHANNEL ERRORS THEN LAYER FAILURE ELSE DOUBLE FAULT	IF NO INTERCHANNEL ERRORS THEN SOURCE_GPC FAULT ELSE DOUBLE FAULT
2 OR 3	IF NO INTERCHANNEL ERROR OR INTERCHANNEL ERROR POINTS TO SAME CHANNEL AS DOES INTRACHANNEL ERROR THEN ICIS FAULT ELSE DOUBLE FAULT	DOUBLE FAULT	DOUBLE FAULT

Figure 4-20. Analysis of Inter-Channel and Intra-Channel Votes

4.2.3.2 ICIS_Local_Manager (ILM) Task

The functions provided by this task include (1) managing responses to detected error conditions, (2) managing responses to updated network layer status received from the Network Manger, (3) providing the capabilities to re-initialize an ICIS, and (4) managing the local retry self tests associated with the ICIS hardware. The task is executed both on an "on-demand" basis in response to an event being set by one of several other tasks requesting services, and on a periodic basis when there are self tests to be executed periodically to check if a previously failed resource has recovered. The ILM task can be activated on demand for the following reasons:

1. the ICIS_RM task reports the detection of a new error condition,
2. the FTP redundancy management software requests that an ICIS be re-initialized,
3. a message has arrived from the Network Manager task.

These relatively diverse functions could have been partitioned into separate tasks but were not due to considerations of minimizing the system's tasking overhead and complexity. As implemented, the task's body is fairly simplistic. The four main functions provided by this task have been individually encapsulated within separate procedures. The main loop of the

ILM task includes a call to each of these procedures. Each procedure checks to see if its function requires action at the current time and, if so, it does what is needed before returning. Each function is given an opportunity to execute each time the task is scheduled to execute.

4.2.3.2.1 Managing Responses to Failures Reported by ICIS_RM Task

The ICIS_RM task performs fault detection and isolation functions on the basis of the received communication packets that it processes. In response to a detected error condition, it must immediately make the responses required to permit continued processing of subsequent input packets (e.g., mask out data from a faulty ICIS). However, there are other less time-critical responses which can be deferred. The ICIS_RM task notifies the ILM task of a new fault condition by calling the ICCS_RM_Failure_Report procedure which provides the inter-task communication and synchronization support required to allow the ILM task to manage these deferred responses. The Check_for_Failure_Report procedure is called from within the main ILM task loop to see if there are any new failure reports enqueued for processing. This procedure manages this processing which includes any further fault isolation functions, fault logging, and scheduling of retry self tests.

Process: ICCS_RM_Failure_Report

Input:

Fault isolation information from ICIS_RM task

Output:

Entry in queue of fault reports

Description:

This process provides a procedure to be called by the ICIS_RM task when it needs to pass new fault isolation information to the ICIS_Local_Manager task. The information is enqueued in a circular buffer structure and an event is set which will result in the execution of the ILM task as some later time. The higher priority ICIS_RM task will not be blocked by the inter-task communication process.

Process: Check_for_Failure_Report

Input:

Fault isolation information from ICIS_RM task

Output:

Error logging information (not currently implemented)
Additional entry in list of requested self-tests to be executed
Conditionally, a message to Network Manager if layer failed

Description:

This process is responsible for dequeueing any failure reports posted by the ICIS_RM task as a consequence of error detections made while processing input packets. Responses to the failures are enacted here with less priority than in the actual ICIS_RM task which has the more time critical responsibility of processing input packets as quickly as possible. The check for new failure reports consists of a comparison of two indices which are part of a FIFO structure used for inter-task communication of the reports.

The enacted responses are conditional on the fault isolation information passed in the failure report. In response to a layer failure, a self test is executed to determine whether the fault is located between the FTP's ICISes and the root node for the layer or is located elsewhere in the layer. This finer localization of the fault is important in that a fault in the ICIS-root node interface can not be circumvented by an intervention of the Network Manager. On the other hand, the Network Manager may be able to correct a fault located elsewhere in the layer by switching inter-nodal links. The self test used to differentiate these two fault conditions involves the execution of an ICIS instruction chain which sends a message to the root node on the layer in question asking the node to respond with a status message. The returned status message is checked for syntactical correctness. A valid status message reception is taken to mean that the root node interface is fault free and it is assumed the layer problem is located elsewhere and will be handled by the IC Layer Manager for that layer. A fault in the root node interface is assumed if no valid status message is returned. In this case, an entry is added to a queue of self test requests managed by the ICIS_Local_Manager task which executes the retry self tests to determine when faults associated with the local ICIS hardware have gone away. This request will be for a periodic check of the root node interface.

In response to fault isolation information passed in the failure report indicative of a total ICIS or a partial ICIS fault, an entry is made requesting retry self tests to determine when the particular type of fault is no longer present. The retry self tests for these types of failures also consist of attempts to "echo" transmissions off of the root node in attempt to determine if the local ICIS hardware is capable of transmitting and receiving on a specific layer or layers. Refer to Section 4.2.3.2.4 for a description of the retry self-test process, the types of self tests executed for the various fault conditions, and the criterion used to determine when a total or partial ICIS fault is recovered.

If the list of retry self tests happens to be empty when this new entry is added, a new task scheduling procedure call is made to the run-time system causing the ILM task to be scheduled for execution on a periodic basis. The task will be run at least every 2 seconds when there is a retry self test to be run. Execution of the task on an "on-demand" basis via the setting of an event remains in effect at all times.

4.2.3.2.2 Managing Layer Status Updates from Network Manager

Process: Check_For_Layer_Repair_Report

Input:
Message from Network Manager received via ICCS

Output:
Updated layer fault status

Description:

This provides an ILM task entry point for receiving communicated messages from a Network Manager indicating the recovery of a previously failed layer. This process is in the form of a procedure which is called on each iteration of the main ILM task loop to check for an updated layer status. Since the location of the Network Manager can be on any FTP in the distributed system, ICCS must be used for inter-task communication. An initial check is made to the ICCS interface available for polling for new messages. If a new message is available, it is evaluated and the status variable maintained by the ICIS RM software reflecting the fault status of each layer is updated if the Network Manager says that a layer has been repaired. The updated status variable will immediately affect all ICIS RM operations (e.g., upon which layers data is expected to be received).

4.2.3.2.3 Managing Requests for Re-initialization of ICIS Hardware

The software responsible for managing the redundancy associated with the core FTP needs to be able to command the re-initialization of the ICIS hardware following the restoration of a channel which has for whatever reason been removed temporarily from the configuration of channels composing the FTP. The re-initialization and alignment of the state of the ICIS hardware could be done within the context of the task managing the re-alignment of the core FTP hardware. However, it was decided not to include the additional time required to initialize this ICIS hardware in the total channel re-alignment time, since the channel re-alignment is done while application tasks are suspended. Furthermore, the time to do the ICIS re-alignment is not time deterministic, as will be shown in the following description of the Align_ICIS process. Therefore the re-alignment is deferred until the lower priority ILM task can process the ICIS alignment request made by the core FTP redundancy management task responsible for managing the re-initialization of FTP channels.

Process: Request_ICIS_Align

Input:
Identification of which channel's ICIS is to be initialized

Output:
ICIS alignment request

Description:

This process enqueues a request by the core FTP redundancy management software to have the ICIS of a particular channel re-initialized. This process is in the form of a subroutine to be called directly by the FTP redundancy management software. The subroutine hides the implementation of the inter-task communication mechanism from the calling task. The subroutine sets an event which will invoke the lower priority ICIS_Local_Manager task to do the actual ICIS alignment when it eventually gets a chance to run.

Process: Check_For_Align_Request

Input:

Request to align the ICIS of a specified channel

Output:

None

Description:

This process is called by the main loop of the ILM task body and checks to see if there are any outstanding requests to align an ICIS. When needed, the process dequeues any available requests made from the FTP redundancy management software to have the ICIS of a particular channel re-initialized. The Align_ICIS process is invoked to actually perform the hardware re-initialization.

Process: Align_ICIS

Input:

Identification of ICIS to align

Output:

Update to variable indicating alignment status of ICISes
Index of "current" buffer used for receiving input packets

Description:

This process is responsible for bringing a specified channel's ICIS into an aligned state with a currently aligned ICIS or ICISes such that all ICISes will synchronously receive and transmit data on the IC network. Much of the state of the ICIS hardware is statically initialized. The more problematic state variables, in the context of re-alignment, are those which dynamically change during the course of receiving and transmitting data. One approach to this problem would be to stop all ICIS operations and do a complete alignment (i.e., replacement of state variables with voted exchange values) of all ICIS state variables

in a fashion similar to the alignment of a channel's processor. However, it is desirable to minimize the disruption of normal ICIS communication activities imposed by the alignment operation and the alignment of the complete state of an ICIS would take tens of milliseconds in its current hardware implementation.

The approach taken to this alignment problem is to align the static variables of the ICIS totally asynchronously to the operations being performed by the ICIS or ICISes already aligned. This alignment process takes place within the context of the ICIS_Local_Manager task while the reception of input packets takes place within the context of the ICIS_RM task, and the execution of ICIS operations to transmit data is performed within the context of the task sending the data out on the network (e.g., the Message Send-Receive task). Once the static variables have been initialized, the dynamic variables are initialized while the ICIS is in a known quiescent state - no data is coming into the ICIS and it is not executing instructions to transmit data. Also, the number of dynamic state variables to align is minimized. Instead of aligning all of the buffer space allocated for storage of the raw, redundant input packet data, only the "unsolicited chain pointer" which points to the current ICIS instruction of the chain of instructions managing the reception of input data and the Last_Pack variable in the ICIS dual-port memory which indicates the last input packet received are aligned. This scheme requires coordination between the ICIS_RM task, which is processing the input packets, and the ICIS_Local_Manager task, which is performing the alignment, so that the resulting non-congruent data across ICISes is not misinterpreted as a fault indication. The Align_ICIS process provides information available to the input packet processing software to determine which input packets are expected to be congruent and which are not necessarily congruent because they were received before the alignment of all currently aligned ICISes. It should be pointed out here that there are other dynamic state variables associated with arbitration for the network and executing solicited chains of instructions (e.g., the state exchange voter masks involved with the inter-channel exchange of states of the polling sequence). These variables, however, are initialized "on the fly" each time a new network polling sequence is started before the execution of a solicited chain and do not require alignment here.

It must be guaranteed that the alignment of even these dynamic state variables related to the reception of data does not occur at a point in time just as new input packets arrive. The variables are aligned during a short window of time following the successful acquisition of the network by the FTP in which the ICIS alignment is taking place. Network possession guarantees that no other FTP in the system can be transmitting data to this FTP. The acquisition of the network is made in the normal manner, i.e., by instructing the ICIS hardware to go through a polling sequence and then execute a specified chain of instructions. (This polling operation is not time-deterministic and introduces variability into the total time required of the Align_ICIS process.) The ICIS instructions cause a continuous stream of "flag" characters to be sourced on the network and thereby assure continued network possession. (The IC network protocol is such that a 512 microsecond period without network activity is an indication that the network is "idle" and therefore a

new arbitration sequence can begin. The current implementation of the ICIS alignment software will always guarantee that the amount of time required to possess the network to perform the alignment of the dynamic variables is much less than 512 microseconds and the "flagging" is not necessary. However, the amount of code executed during network possession can be reduced and this implementation will continue to work.) The minimal, critical alignment code is executed while the FTP has network possession; then the FTP immediately releases the network allowing any waiting sites to begin a new arbitration sequence.

Before the overall ICIS alignment process is started, a check is made to see if the channel in which the specified ICIS resides is currently in synchronous operation with the majority; the ICIS alignment is only attempted if the channel is capable of synchronous operation. The initialization of the static state variables proceeds in four steps:

1. The chain of ICIS instructions required to execute while the FTP maintains network possession such that dynamic state variables are aligned is initialized in the ICIS dual-port memory. This simple chain consists of an infinite loop in which SDLC "auto flags" are turned on.
2. The chain of ICIS instructions required to execute the ICIS "retry" self tests is initialized in ICIS dual-port memory.
3. The chain of ICIS instructions forming the unsolicited chain used while the ICIS is in unsolicited mode and supports the reception of input packets is initialized.
4. The ICIS hardware control registers of the ICIS being aligned are initialized such that the ICIS can begin to execute the unsolicited chain of instructions and can begin to receive input packets. Note that the registers are loaded with constant initial values and not with values resulting from a voted exchange among the channels.

If the ICIS being aligned is the only ICIS available in the FTP either because this is a simplex processing site or because of massive ICIS failures at a non-simplex site, the ICIS alignment process concludes at this point with the starting of the execution of the unsolicited chain used to receive packets.

Otherwise, the FTP's ICIS complex must get network possession, align the dynamic state variables, and bring the ICIS into synchronous operation with the already aligned ICIS or ICISes (i.e., it will place new input packets in the same dual-port memory buffer and will have the same Last_Pack value to indicate which was the last packet received). The aligned ICIS is queried to determine when it is not executing a solicited chain. When available, the sequencer is directed to poll for network possession and to execute the "dummy" null loop chain. After the command is issued to the sequencer to poll for network possession, the IOP must begin a poll of the ICIS Chain Status Register to determine when the ICIS actually has obtained network possession. Once network possession is confirmed, the unsolicited chain pointer from the ICIS or ICISes already aligned is copied into the chain pointer for the ICIS being aligned. A similar alignment operation is made for the

Last_Pack variable along with a recording of the variable's current value to be used by the ICIS_RM task for determining when the newly aligned ICIS should have congruent input packets. A status variable is updated to indicate that the new ICIS is officially aligned. The processor then writes directly to ICIS control registers to shut off the "auto flagging" activity, to release the network, and to transition the ICIS back to unsolicited mode. Note that the execution of this "alignment" solicited chain is no different than the execution of any other chains and that all the error detection checks associated with executing a chain (e.g., check for time outs waiting for the sequencer to become available to execute a chain) are made.

4.2.3.2.4 Managing Retry Self-Tests

It is assumed that the detected error conditions associated with the IC hardware local to a FTP (e.g., total ICIS, inter-ICIS links, interface between ICIS and layer root node) may be transient in nature. The benefit of being able to recover from transient faults is increased communication reliability. Retry self tests have been developed to determine if a previously failed resource is usable at some later point in time. Following the recording of a fault by the ILM task, the faulty resource is retried periodically via one of these self tests to determine if it is again fault-free. If the test results indicate that the resource is fault-free, it is again used for normal communication activities. Note that the implementation supports multiple, simultaneously outstanding faults in the IC hardware complex local to a FTP.

Process: Run_ICIS_Retries

Input:

List of records indicating current faulty ICIS resources

Output:

Modification of status variables if resource recovered

Description:

The Run_ICIS_Retries process manages the self-test retries to determine when a faulty ICIS resource has recovered. The implementation supports the management of multiple faulty resources at the same time. The term "resource" refers to a fault isolation region of either a total ICIS, a partial ICIS, or the interface with the root node. Recovery of a layer fault is managed by the Network Manager.

The execution or retry self-tests associated with a particular fault are performed at time intervals which are exponentially lengthened as the self-tests fail to indicate a recovery. That is, less and less processor and ICIS throughput is dedicated to checking on the recovery status of a resource as time goes on following the initial error detection. The current implementation has the initial retry interval set for 2 seconds and the maximum retry

interval limited at 5 seconds. The operations performed during the course of a self-test, the criterion for recovery, and the responses made to a recovery all depend upon the fault isolation region associated with the fault, i.e., total ICIS, partial ICIS, or root node interface. In all cases, an entry in the list of outstanding faults is maintained for the particular fault until it is deemed recovered. If a recovery is detected for the one and only outstanding fault recorded in the list of faults maintained for managing the retry self tests, the task scheduling parameters for the ILM task are updated such that the task is no longer scheduled on a periodic basis but only "on-demand."

Total ICIS Fault

The retry self-test associated with a total ICIS fault is executed only if the corresponding channel is in synchronous operation . The ICIS to be tested is first initialized using the standard Align_ICIS process. For each of the three layers on which it is expected that this FTP can both transmit and receive, a self-test solicited chain is executed. Each chain executes the appropriate OUTPUT and INPUT instructions for getting status from a layer's root node to the FTP. Each channel's ICIS should receive a copy of the node's return status. Each redundant status packet is individually validated and indicates which channel's ICIS passes the test. The criteria for considering a total ICIS recovered is:

1. the ICIS can transmit data to and receive data from the root node on the layer which it, the ICIS, physically transmits on, and
2. the ICIS is at a simplex FTP site and it can pass the self test on at least one layer, or it is not a simplex site and it can pass the test on more than one layer.

When these criteria are met, the recovered ICIS is again aligned and its "health" status variable is changed to non-faulty; the ICIS will be included in other ICIS RM operations such as receiving and transmitting data.

Partial ICIS Fault

Here it is assumed that a fault is localized to only an ICIS subsection associated with the reception of data on a single layer, or the fault may be associated with the inter-ICIS links used to relay layer data from one ICIS to another in the same FTP. The retry self test consists of the transmission of a status request message to the root node on the affected layer. If only one ICIS is affected by the fault, it must be able to receive a valid status message from the root node before the ICIS is considered to be recovered. If two ICISes are affected, both must pass the self test before a recovery is declared. The recovery response made depends upon which type of reconfiguration response was made by the ICIS_RM task when the fault was initially localized. Either the ICIS involved with the fault was marked as not usable or the layer was marked as not usable. The type of initial reconfiguration response is recorded in the fault entry on the list maintained by Run_ ICIS_Retries. If the ICIS was made unavailable it is reinstated after an alignment operation

is performed. If the layer was made unavailable, it is again made available by updating the appropriate status variable.

Root Node Interface Fault

The retry self-test for a root node interface fault consists of an attempt to transmit a "give-me-status" request to a single layer's root node and then attempt to receive a valid status packet from the node. If any channel's ICIS successfully receives a valid return status packet, then the interface fault is judged to be recovered. The status variable associated with the health of layers is updated when a recovery is detected; the layer will be included in all ICIS RM operations once again.

4.2.3.3 Starting Solicited Chain

The ICIS hardware redundancy is not completely transparent during the processes of arbitrating for network possession and the execution of instruction chains in the solicited mode. ICIS control registers and a layer redundancy status value in all communicated packets must be dynamically updated just before every network arbitration and solicited chain execution sequence. The registers and status value are updated on the basis of the currently known layer, ICIS, and channel redundancy levels. This redundancy management support is required for fault-free IC communication.

This redundancy management software also provides error detection functions. ICIS status is analyzed in an attempt to detect faults associated with network arbitration and the execution of solicited chains.

Process: Start_Solicited_Chain

Input:
Address of solicited chain to be executed

Output:
Return status indicating error detection results

Description:

The Start_Solicited_Chain process is multi-faceted in terms of the required functions that it supports. It manages the hardware redundancy involved in arbitrating for the network and in executing solicited chains. It performs error detection functions related to the execution of solicited chains where data is transmitted on the network. It provides a limited amount of fault recovery functionality in the sense that it will invoke the "possession default recovery" process when a possession default state exists. And finally, the process synchronizes the attempts made by multiple tasks to execute solicited chains.

This process is embodied in a procedure with one input parameter - the address of the solicited chain to be executed. The procedure is called after the solicited chain and any output packet data have been written in the ICIS dual-port memory. After performing a range check of the start address parameter, the process of executing the chain is begun. A loop of code is executed repeatedly with each iteration consisting of three steps: (1) the determination of the appropriate values for certain ICIS control registers, (2) a read of the Chain_Status_Register (CSR) to determine if there is still a solicited chain in progress, and (3) a conditional updating of control registers and the commanding of the ICIS sequencer to execute the solicited chain if the previous chain has completed. The CSR value read also contains bits of information indicating the presence of possession_default, poll_tx_fail, and data_tx_fail error conditions. These error bits are checked on each loop iteration. Execution of the loop continues until:

1. the new chain can be started with no error conditions noted, or
2. an error associated with the previous solicited chain execution is detected (will still attempt to start this new chain), or
3. a time limit expires before the previous chain completes.

Each iteration of the Start_Solicited_Chain loop consists of the following operations:

1. A determination must be made as to which redundant ICISes will be instructed to execute the chain and from which set of ICISes read accesses will be made to collect status. Aligned, fault-free ICISes from channels reported to be in synchronous operation by the FTP redundancy management software are included in this process.

2. A new value to be loaded into the Poll_Priority (PP) ICIS register is calculated as a function of the available fault-free channels, ICISes, and layers. The PP register value defines the sequence of polling bits to be placed on the network when this FTP participates in the network arbitration process. The first part of the poll bit sequence is referred to as the "redundancy contention sequence" in which the network subscribers of differing redundancy levels will arbitrate. One of the polling bits is the Triplex bit which is only placed on the network by triplex network sites. Another bit, the Duplex bit, is used to denote that the source of the poll bit is a duplex site. Thus sites of higher redundancy level will always win a polling sequence at the very beginning of the polling sequence. The Triplex bit in the PP register has additional significance - it determines whether redundant, incoming poll bits monitored on the redundant layers are voted or logically ORed. That is, when the Triplex bit is set the poll bits from the three layers are voted; otherwise the bits are ORed. The PP value will only have the Triplex bit set if all three channels and all three ICISes of the FTP are fault-free and the FTP does not perceive any current layer faults (including faults on the root node interface). The lower level poll priority bits are used to break ties and are always initialized with the particular FTP's unique ID.

3. The voters used for inter-channel exchanges of the ICIS state machine's states have two bits of registered information required to select the masking of the three inputs to the voters. These bits are located as the most significant two bits of the ICIS register located at address 15_{16} in the ICIS address space and are referred to as M_0 and M_1 in the hardware documentation. These control bits are determined on the basis of the current channel configuration, the channel's ID, and the configuration of aligned and fault-free ICISes currently available.

4. The configuration of network layers on which data is to be transmitted must be determined. This information is used to enable the HDLC transmitters linked to particular layers and is used to encode in the outgoing packet the expected layer redundancy of this transmission which is used at the receiving site for layer fault detection purposes. A layer is said to be available for transmission if the ICIS transmitting on it is healthy and aligned and is in a channel in the current configuration (i.e., its monitor interlock is engaged). Also, the layer must be fault-free from the perspective of the transmitting FTP before it is considered available for transmission. Note that this requirement for non-faulty layer status is relaxed if the solicited chain to be executed is part of a retry self-test process.

5. A mask value is created to disable the inclusion of incoming poll-bit information from any layers considered to be faulty.

6. The Chain Status Register (CSR) is read and the Chain_Complete status bit is tested to determine if the ICIS is free to start a new solicited chain; the ICIS may be busy even in a fault-free condition. All interrupts to the processor are disabled while the Chain_Complete status check is made and the new solicited chain is started. This section of code is a "critical section" and can not be preempted by another task which may happen to also want to start its own solicited chain at this same time. The "ICIS solicited chain executer" is a shared resource among the multiple tasks of the system and access to it is protected by the Chain_Complete bit. The starting of a new solicited chain involves the following steps:

    1. Load the Poll Priority register
    2. Load the ICIS "Location 15" control register
    3. Write the layer redundancy encoding in output packet
    4. Set the Solicited_Chain_Pointer to start of new chain
    5. Set the Interface_Control_Register (ICR) with value to poll for network possession and execute solicited chain

    The Chain_Complete bit is cleared upon the write to the ICR. The processor interrupts are re-enabled.

A copy of the sampled CSR register is analyzed after each attempt to start the solicited chain. The bits in the CSR associated with ICIS error conditions are checked and a check is also made as to the time which has elapsed while attempting to execute a new chain. If an error condition is detected, a response is made immediately (e.g., attempt to recover from a possession default condition) and the error condition is reported back to the subroutine caller in the form of a return value.

Process: Possession_Default_Recover

Input:
None

Output:
None

Description:

This process performs a reset function on the ICIS state machine which releases it from being infinitely stuck in the possession_default state and from never being able to poll for network possession again. All transmitter outputs to the network layers are turned off just in case the ICIS is "babbling." The state machine is reset by setting the STOP bit in the Interface Control Register and then clearing the STOP bit. The state machine should then reset to the Not_polling state and be ready to respond to the next request to poll for network possession.

4.2.3.4. Process to Check the Status of the ICIS

A subroutine is available which checks the current status of the ICIS and returns a status indication to a caller. This subroutine checks for possession default conditions and checks for stuck layer conditions (i.e., a network layer's physical signal level remains high for more than 512 microseconds). If a possession default condition is detected, the possession default recovery process is invoked. If a stuck layer condition is detected, the layer's status is changed to a faulty state and the appropriate Network Manager is notified.

5.0 THE NETWORK LAYER

To permit inter-computer communication in the AIPS distributed system, a data communication mechanism was developed. This facility uses high-speed links to attain high throughput and hardware redundancy to achieve fault tolerance. Additionally, it employs a suite of software modules to implement the *Network Layer* of the ISO Model, thus enabling abstracted inter-process communication. The Network Layer is responsible for creating a virtual circuit, providing a standard interface to this path, and hiding the complex mechanisms of its operation from the higher layers of software [4]. This chapter describes the Network Layer that was designed for the AIPS Distributed Engineering Model.

The AIPS *IC network* consists of three identical independent IC layers (not to be confused with the ISO Layers) which operate in parallel to provide reliable communication and to dynamically mask faults (illustrated in Figure 5-1). To allow inter-computer communication, a virtual path is routed, or grown, between the distributed computers. This path is constructed by the *IC Network Manager*, which is part of the *IC Communication Services*. The IC Communication Services is a set of processes that support inter-computer communication between fault tolerant FTPs of varying redundancy. The IC Network Manager is the process which grows and maintains the AIPS IC network. Since the AIPS Distributed Model utilizes three independent layers, the IC Network Manager is composed of three *IC Layer Managers*. Each Layer Manager is responsible for detecting, isolating, and reconfiguring around hardware faults in its respective network layer. Additionally, to enable a completely distributed system, the three Layer Managers may reside on different FTPs.

Sections 5.1 and 5.2 provide the functional requirements and design for the IC network growth and FDIR respectively. Similarly, Sections 5.3 and 5.4 present the software specifications for the IC growth and FDIR.

5.1 IC Network Growth Functional Requirements and Design

To enable fault-tolerant communication between the AIPS distributed processors, a redundant set of communication paths was constructed. These paths are referred to as the Inter-Computer Network, and the process that leads to their creation is called the IC Network Growth. As stated earlier, the growth of the IC network is performed by the IC Network Manager.

AIPS INTER-COMPUTER(IC) NETWORK

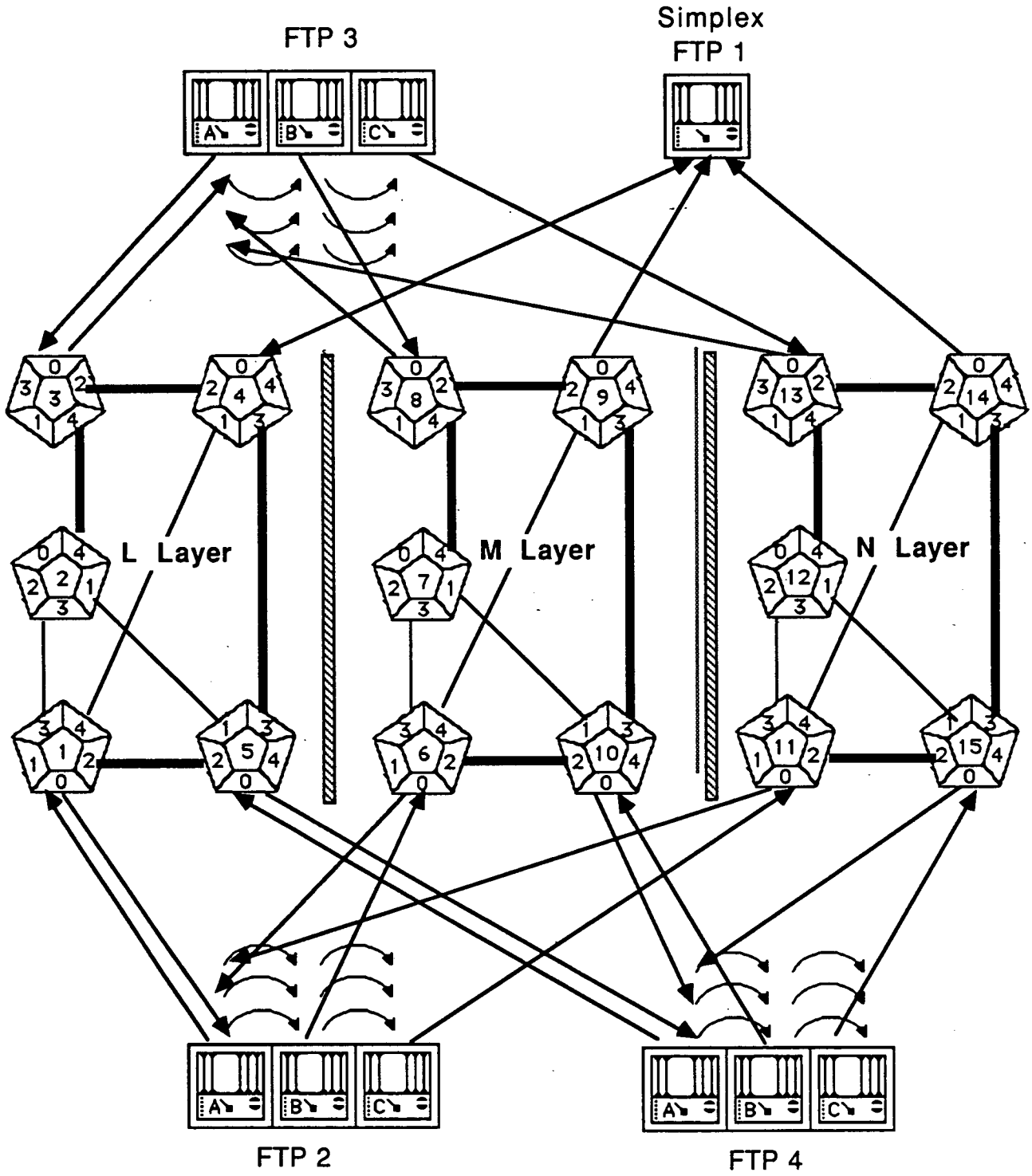


Figure 5-1. AIPS Inter-Computer Network

To permit a dynamically reconfigurable system, each triplex (or higher redundancy) FTP may attempt to grow the IC network. Although multiple FTPs are capable of performing the growth, only one FTP successfully completes it. That is, if *two* or more triplex FTPs simultaneously try to grow the IC network (contend for the growth), then *one* or more of these FTPs is "backed off" (suspended). The IC Network Growth Algorithm ensures that a virtual communication path is constructed, even if multiple FTPs contend for the growth.

The IC Network Manager and the IC Network Growth Algorithm are described in this section.

5.1.1 Overview of the IC Network Growth

The growth of the IC network entails the creation of a virtual path allowing the communication of information between the FTP sites. In steady state, the communication path operates as a time division multiplex bus. This bus differs from a conventional linear bus in that data is routed by circuit switched nodes through one of several possible paths (depicted in Figure 5-1). The use of circuit switched nodes allows spare interconnections, which can be brought into service if a hardware fault occurs (or a network component is damaged). This network architecture provides coverage for many failure modes which cause a standard linear bus to either fail completely or provide service to a reduced set of subscribers.

Data flow in the IC network is controlled by the configuration of the ports in each node. For a link to carry data between two nodes, the ports at either end of the link must both be enabled. Nodes retransmit messages received from an enabled port on its other enabled ports, but not on the port that received the message. When a node receives a message addressed to itself from any port, disabled or enabled, it carries out the command encoded in the message and then transmits its status on all of its enabled ports, including the port which received the message if that port is enabled. A node obeys the reconfiguration commands sent by the IC Network Manager by enabling or disabling its ports in accordance with the value of the command's port enable field. Once the new configuration is in effect, the node returns a status message. For proper operation, there can be no loops in any layer. Accordingly, a data bit travels through each enabled link exactly once.

The IC Network Manager performs the growth of the IC network by executing a series of node reconfiguration and status chains. The *reconfiguration chains* instruct one or more nodes to configure their ports. Alternatively, the *status chains* merely query a set of nodes for their status. These chains are used to simultaneously grow each layer of the IC network. To permit this coincident growth of the IC layers, each reconfiguration chain utilizes three transactions, one transaction per layer. A *transaction* is an autonomous command to configure a node's ports or request a node's status. A *reconfiguration transaction* is a command that configures a node's ports such that the node is added to the

IC virtual path. After the designated node processes the reconfiguration transaction, it returns its status. If the transaction is executed successfully, then a fault free response from the node is returned.

If the reconfiguration chain and corresponding status chains execute without any errors, then the associated nodes are added to the IC network. If one or more transactions in a chain has an error, then either contention for the network growth has occurred or faults exist in the IC network (the details concerning the differentiation of network contention from hardware faults are discussed in Sections 5.1.4 and 5.1.5). If network contention occurs, then IC Network Managers on the contending sites are backed off. Alternatively, if faults exist, then the IC Network Manager modifies the reconfiguration chain to bypass the faults and then re-executes the chain.

The virtual path in each layer is constructed incrementally. The details concerning the algorithm used to select the next node to add and the corresponding route are described in Section 5.3.6, IC Network Growth.

After the IC Network is grown, the *System Manager* is assigned to a particular FTP. The System Manager is a collection of functions that allocate migratable functions to the FTPs, supervise system FDIR, and maintain a consistent time. To simplify the IC Network growth and the subsequent assignment of the System Manager, several restrictions are currently imposed:

1. The IC Network Manager that performs the IC network growth assumes the network FDIR responsibilities for all layers. That is, the IC Layer Managers are allocated to the same FTP as the IC Network Manager that grows the IC network. Accordingly, these Layer Managers respond to all IC network FDIR requests.

2. Any IC Network Manager that does not perform the growth of the IC network is an *Alternate IC Network Manager*. An Alternate Manager does not respond to an IC network FDIR request unless the FDIR function is migrated to it.

3. The FTP that performs the IC network growth assumes the System Manager responsibilities.

4. A triplex (or higher redundancy) FTP may attempt to grow the IC network only if it can communicate on all three layers of the network. Loss of communication to a layer can be due to a faulty root node, ICIS, channel, etc.

5.1.2 Initialization of the IC Network

The IC Network growth is a small part of the IC network initialization. In brief, this initialization process involves the activation of the local FTP processes (or *Local System*

Services), the actuation of *IC Communication Services*, the growth of the IC network, the assignment of the System Manager, and the completion of network diagnostics.

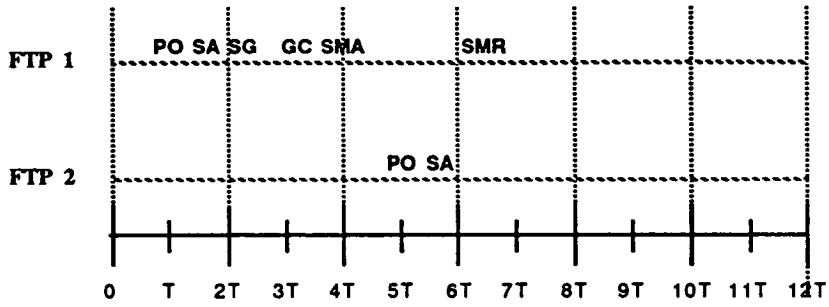
When an FTP is powered up, the Local System Services are started. These Services are responsible for synchronizing the FTP channels and initializing the required local tasks. After the local tasks are started, the IC Communication Services are activated. The IC Communication Services provide the Session and Transport layers of the ISO Model, and they are started to support the initialization the IC Network. During this initialization phase, these Services are used to determine whether or not the IC network has been grown.

To decide if the IC network has already been grown, the local *FTP Resource Allocator* broadcasts a "Site is Accessible" (SA) message to all remote sites (using the IC Communications Service). The FTP Resource Allocator is a Local System Services process which coordinates and manages any global or migratable functions that are assigned to the FTP. The Resource Allocator sends the SA message to indicate that this FTP has completed the initialization of its Local System Services.

After the FTP Resource Allocator transmits the SA broadcast, it waits for a response from the System Manager for a predetermined period of time (period > worst case IC network growth time + IC communications overhead). This response will be either a specific acknowledgement to the SA message or a general "Network is In_Service" (NIS) broadcast. The "Site is Accessible" acknowledgement is sent by the System Manager if the Manager was activated before the SA message was transmitted. The System Manager broadcasts the SA acknowledgement to all FTP Resource Allocators to inform them that the IC network is in_service. This acknowledgement is composed of: the FTPs that have previously completed initialization, the redundancy levels of these initialized FTPs, the location of the System Manager, and any updated status information. The "SA message, SA acknowledgement" scenario is illustrated in Figure 5-2.

Alternatively, the "Network is In_Service" message is broadcast by the System Manager and received by the FTP Resource Allocator, if the Manager was assigned after the SA message was issued but before the SA time out had expired. If the FTP Resource Allocator receives the NIS message, then it re-broadcasts its SA message. The SA message is re-sent, because the System Manager did not receive the first transmission. The "SA message, NIS message, SA message" scenario is depicted in Figure 5-3.

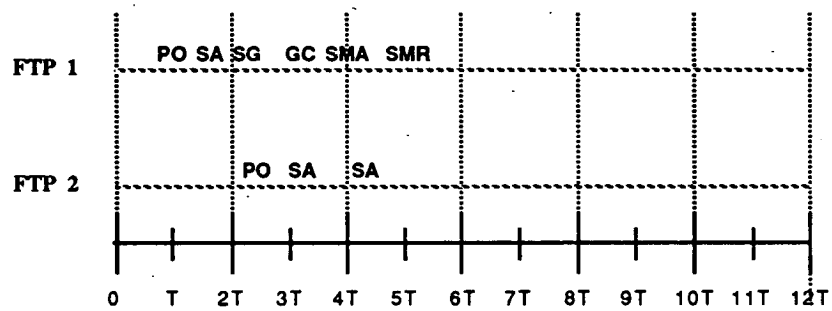
If neither the "Site is Accessible" nor the "Network is In_Service" message is received by the Resource Allocator before the SA time out expires and the FTP is at least a triplex, then the Resource Allocator requests that the resident IC Network Manager attempt to grow the network. In contrast, if the time out expires and the FTP is a simplex or duplex, then the FTP Resource Allocator continues to wait for the "Network is In_Service" message.



LEGEND:

PO - Power On
SG - Start Growth
SMA - System Mgr. Assigned; Net. is In_Service Broadcast
CBO - Contention and Back Off
RG - Attempt to Regrow Network
SA - Site Accessible Broadcast
GC - IC Growth Completed
SMR - System Mgr. Responds to SA
NDC - Net. Diagnostics Completed
MFG - Multiple Layer Fault Occurred during Growth
MBG - Multiple Layer Fault Exists before Growth
T - Worst Case Growth Time

Figure 5-2. IC Network Initialization - FTP Accessible After System Manager Assignment



LEGEND:

PO - Power On
SG - Start Growth
SMA - System Mgr. Assigned; Net. is In_Service Broadcast
CBO - Contention and Back Off
RG - Attempt to Regrow Network
SA - Site Accessible Broadcast
GC - IC Growth Completed
SMR - System Mgr. Responds to SA
NDC - Net. Diagnostics Completed
MFG - Multiple Layer Fault Occurred during Growth
MBG - Multiple Layer Fault Exists before Growth
T - Worst Case Growth Time

Figure 5-3. IC Network Initialization - FTP Accessible Before System Manager Assignment

When resident IC Network Manager endeavors to grow the IC Network, either of two outcomes is possible: (1) the Manager completes the growth, or (2) another FTP contests the growth and this Manager backs off. If the IC Network Manager backs off, then, after the back off period expires, it performs the the same initialization protocol that the FTP Resource Allocator previously initiated. Specifically,

1. The IC Network Manager broadcasts an SA message. This message is sent to determine if the IC network was grown while the FTP was backed off.

2. The IC Network Manager waits for a response from the System Manager for a predetermined period of time. This response will be either the SA acknowledgement or the NIS broadcast.

3. If neither response is received before the SA time out expires, then the IC Network Manager re-attempts to grow the network.

4. If the SA acknowledgement is received, then the IC Network Manager informs the FTP Resource Allocator that the IC network is in_service.

5. If the NIS message is received, then the IC Network Manager re-sends the "Site is Accessible" message and waits for the subsequent acknowledgement.

5.1.3 Assignment of the System Manager

After an IC Network Manager completes the growth of the IC network, it notifies the resident FTP Resource Allocator that the network is in_service. Since this FTP completed the IC network growth, it assumes the System Manager responsibilities. Furthermore, the IC Network Manager notifies the System Manager of the presence and location of any network faults.

After the System Manager has been actuated, it uses the IC Communications Service to broadcast the "Network is In_Service" message. In addition to informing all FTPs that the network is in_service, this message specifies the location of the System Manager and any updated status information. Subsequently, the System Manager delays for a pre-determined time waiting for any "Site is Accessible" messages sent by remote FTPs. When it receives an SA message, it responds by broadcasting an acknowledgement to inform all FTPs that the FTP which transmitted the SA message is available. After the NIS time out has expired, the System Manager updates its FTP configuration database.

After all FTPs are accessible, a suite of IC network diagnostics is performed by the IC Network Manager. These diagnostics attempt to detect hardware faults in the spare network components and recheck any failed nodes or links. After the diagnostics are

completed, the IC User Communications are enabled permitting inter-processor transmission between application tasks.

5.1.4 Network Contention

When two or more FTPs simultaneously attempt to grow the IC network, network contention occurs. As stated earlier, these contending sites are backed off and may attempt the growth at a later time. Additionally, to complicate the IC growth process, hardware faults may also exist in the network. Consequently, the algorithm must be capable of distinguishing between FTP contention and network faults.

The occurrence of network contention can be easily detected if either of the following two assumptions is made.

1. The IC network is fault free.
   - If the network is guaranteed to be fault free during the IC network growth, then all chains will execute without errors unless two or more FTPs simultaneously attempt to grow the network. As a result, if a transaction in a reconfiguration or status chain has an error, then network contention has occurred.

2. A fault can only exist on a single layer.
   - If faults are restricted to one of the three layers of the IC network, then all chains will execute with a maximum of one error (these chains employ one transaction per layer) unless two or more FTPs simultaneously attempt to grow the network. As a result, if a reconfiguration or status chain has two or more errors, then network contention has occurred.

However, if assumptions are *not* made concerning the existence or position of hardware faults, then it is sometimes difficult to detect the occurrence of network contention, because contention and faults can generate the same error symptoms. For example, network contention can generate the following error symptoms:

1) The FTP performing the growth of the IC network receives a message other than a node response (such as reconfiguration or status commands from a different FTP).

2) The disconnection of nodes that were previously connected to the IC network.

3) The loss of multiple node responses or the occurrence of errors in multiple node responses.

Nevertheless, these identical symptoms can be caused by the following hardware faults:

1) Babblers retransmitting old command frames.

2) Faults that manifest themselves during the growth of the network.

3) Multiple link or node faults on different layers.

Since the occurrence of network contention and existence of hardware faults can generate similar error symptoms, the growth algorithm can not differentiate between contention and faults based on a single set of error information. Consequently, in order to consistently distinguish network contention from faults, each iteration of the growth algorithm (the attempt to add three nodes to the network, one per layer) must involve *multiple* steps, where each step performs an *independent* task. As a result, if contention occurs or faults exist, each iteration will generate several independent sets of errors (rather than just one set if a single step iteration is used). Accordingly, if the steps of each iteration are chosen correctly, the resulting set of error symptoms will allow the algorithm to consistently differentiate between the occurrence of network contention and the existence of hardware faults.

5.1.5 IC Network Growth Algorithm

During the AIPS power-on sequence, all the layers of the IC network must be grown. The IC Network can be grown using a centralized algorithm, but a fully distributed system should be able to perform a distributed growth. The primary advantages of a distributed growth algorithm are that: a deterministic power-on sequence is not required and the AIPS system can automatically adapt to failures in processing sites.

The fundamental process in the algorithm for the distributed growth of the IC network is the incremental addition of nodes to each layer of the IC network. The incremental addition of three nodes, one per layer, is considered an iteration of the growth. Each iteration of the growth algorithm involves the following steps:

1) Request the status of the nodes that have been previously added to the IC network.

2) Execute a reconfiguration chain to add three target nodes (one per layer) to the IC network.

3) Request the status of the nodes that should be connected to the IC network (the nodes of step 1 + the target nodes).

Step 1 of each iteration is primarily used to detect the occurrence of network contention. This step is a valid means of perceiving contention, because of the method that is used to add the circuit switched nodes to the IC network. Specifically, when an attempt is made to add a node to the network, all ports of the node are disabled except the port through which the connection to the node is made (the node's inboard port). As a result, if two FTPs are simultaneously growing the network, one FTP will eventually disable the ports of a node that another FTP has enabled, thus causing the latter FTP to have an inconsistent view of the network. Accordingly, when step 1 is executed, this latter FTP (or possibly both FTPs depending on the timing) will detect errors and conclude that network contention has occurred.

Step 2 attempts to add three target nodes to the IC network. One node is added to each layer. Further, these nodes may or may not be in the same relative position within each layer (e.g. nodes 1, 6, and 11 in Figure 5-1 are in the same relative position in each layer, whereas nodes 3, 6, and 12 are in different relative positions). If this step is adding nodes that are in the same relative position, then the iteration is performing a "*uniform growth*". Conversely, if the execution of this step is adding nodes to the network that are in different relative positions, then a "*non-uniform growth*" is being performed.

Finally, step 3 is a confirmation step. When executed, it verifies that the nodes addressed in step 1 and the target nodes added in step 2 are still accessible. This step is also used to detect the occurrence of network contention.

After each iteration is executed, the node responses from each step must be processed (the nodes return data in response to the reconfiguration and status chains). These responses are utilized to indicate whether a hardware fault exists, network contention occurred, or the nodes were added successfully. The analysis performed by the IC Network Growth Algorithm is illustrated in Figures 5-4 and 5-5 and is outlined below:

1. If errors do not occur during the execution of all three steps of the iteration, then the associated nodes are added to the IC network.

2. If a single error occurs during the execution of step 1, then the algorithm assumes that a remote FTP is performing a non-uniform growth and has modified the port configuration of a node.

    a. The algorithm presumes that contention has occurred and backs off.

    b. It is possible that the error is due to a node (or link) that failed during the execution of the growth rather than network contention. However, if a hardware fault is the cause of the error, it will be detected during the re-growth of the network (by a remote FTP or this FTP after its back off time has expired).

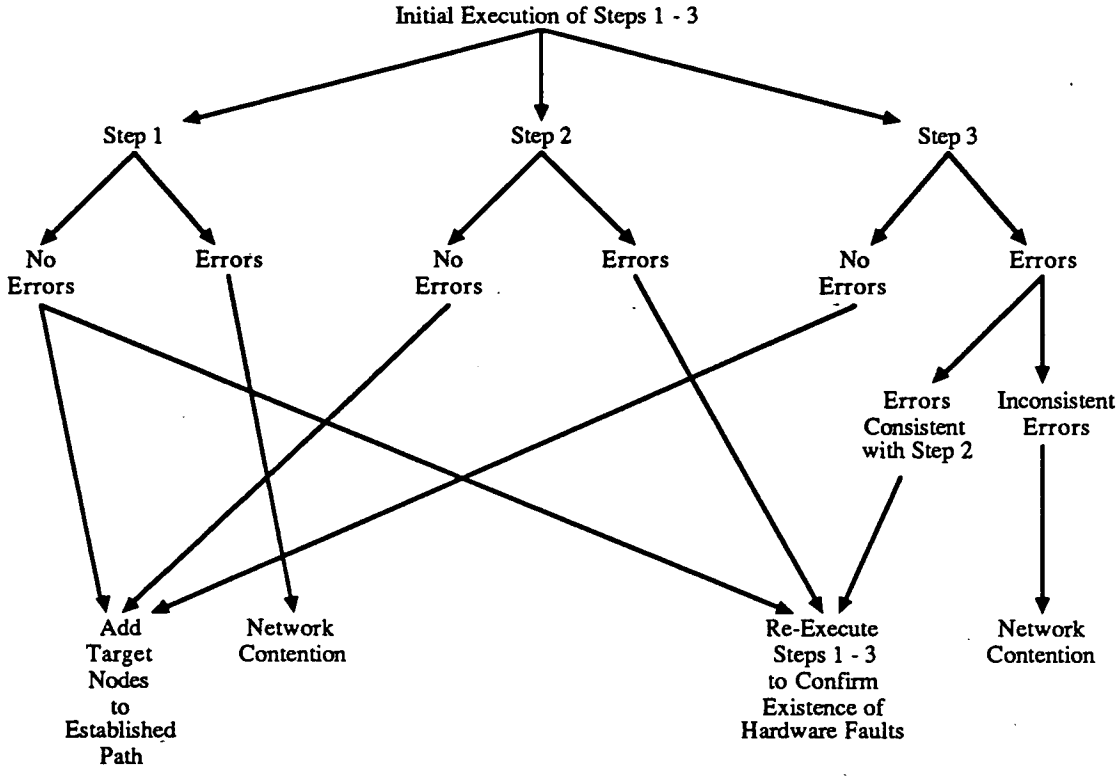


Figure 5-4. Processing of the Execution of Steps 1 - 3

3. If errors occur on two or more layers in step 1, then the algorithm assumes that a remote FTP is performing a non-uniform or uniform growth and network contention has occurred.

    a. The algorithm backs off.

    b. It is possible that the errors are due to a nodes or links that failed during the execution of the growth rather than network contention. However, if a hardware faults are the cause of the errors, it will be detected during the re-growth of the network (by a remote FTP or this FTP after its back off time has expired).

4. If errors do not occur in step 1, one or more errors occur during the execution of step 2, and consistent errors occur in step 3, then steps 1 - 3 are executed again using the same set of chains.

    a. The chains are re-executed to confirm that the multiple errors were due to hardware faults rather than network contention.

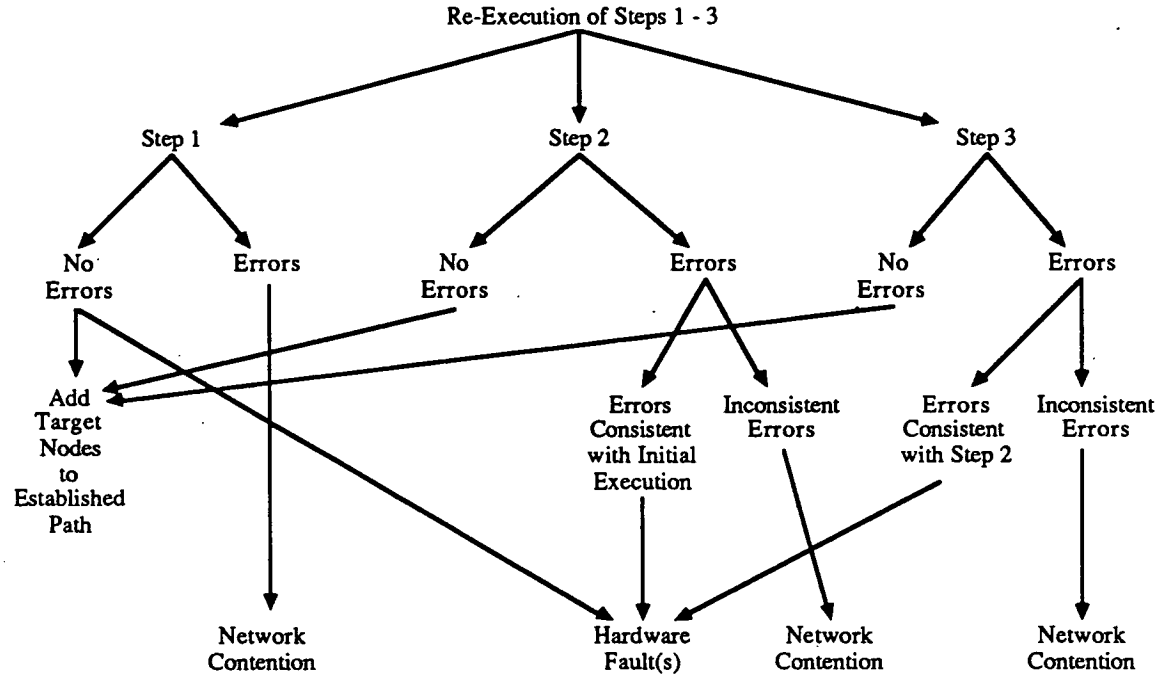


Figure 5-5. Processing of the Re-execution of Steps 1 - 3

b. Each FTP will delay a different non-zero length of time before re-executing steps 1 - 3. This delay is incorporated to ensure that if two or more FTPs that "directly contend" during the first execution of the chains, then they will not contend during the second execution. (In this context, *direct contention* occurs when two or more chains are executed at the exact same time by different FTPs, thus causing the loss of or errors in the expected node responses). The delay that is imposed must be greater than the time required to execute and process steps 1 - 3 (Delay >= FTP_ID * worst case time).

5. If errors do not occur in step 1, errors occur during the execution of step 2, and inconsistent errors occur in step 3, then the algorithm assumes that network contention has occurred.

   - Network contention is assumed, because of the dissimilar data that resulted from the execution of steps 2 and 3.

6. If errors do not occur in steps 1 or 2 and one or more errors occur during the execution of step 3, then the algorithm assumes that a remote FTP is

performing a non-uniform or uniform growth and network contention has occurred.

7. If errors do not occur during the re-execution (re-executions are only performed to verify the existence of hardware faults - case # 4) of steps 1 - 3, then the algorithm assumes that either network contention or transient errors caused the errors in the initial execution of the procedure.

    a. Since the re-execution of the procedure did not have any errors, then the associated nodes are added to the network.

    b. It is possible that the errors which occurred during the initial execution of steps 1 - 3 were due to contention with one triplex FTP and that errors did not occur in the re-execution of the steps because of the different delay between retries. As a result, the procedure will not detect the occurrence of network contention during this iteration of the algorithm. However, as the contending FTPs continue to grow toward each other, the detection of network contention (via errors in step 1 or 3) is inevitable.

8. If one or more errors occur during the re-execution of step 1, then the algorithm assumes that network contention has occurred.

    a. Network contention is assumed, because of the dissimilar data that resulted from the execution and re-execution of the procedure.

    b. It is possible that the errors are due to nodes (or links) that failed during the execution of the growth rather than network contention. However, if hardware faults are the cause of the errors, they will be detected during the re-growth of the network (by a remote FTP or this FTP after its back off time has expired).

9. If errors do not occur in step 1, one or more errors occur in step 2 during the re-execution of of the procedure (errors similar to the errors resulting from the initial execution of step 2), and consistent errors occur in step 3, then the algorithm assumes that hardware faults exist on multiple layers of the IC network.

    a. The corresponding reconfiguration chain is modified to bypass the faults and steps 1 - 3 are executed again.

    b. It is possible that the errors which occurred during the initial execution of steps 1 - 3 were due to contention with one triplex FTP and that the

errors which occurred in the re-execution of the procedure were due to contention with a different FTP. As a result, the procedure will falsely assume that network faults exist. However, this situation will be rectified during the final phase of the growth because all previously failed components are re-tested.

10. If the errors that occur during the re-execution of step 2 are not consistent with the errors that result from the initial execution of step 2, then the algorithm assumes that network contention has occurred.

11. If errors do not occur in step 1, errors occur during the re-execution of step 2, and inconsistent errors occur in step 3, then the algorithm assumes that network contention has occurred.

    - Network contention is assumed, because of the dissimilar data that resulted from the re-execution of steps 2 and 3.

12. If errors do not occur in steps 1 or 2 and one or more errors occur during the re-execution of step 3, then the algorithm assumes that a remote FTP is performing a non-uniform or uniform growth and network contention has occurred.

5.1.6 Analysis of Network Contention Error Symptoms

As mentioned in Section 5.1.4, the motivation behind the development of a *multi-step* growth algorithm is to provide the capability of differentiating between the existence of hardware faults and the occurrence of network contention. In that section, three error symptoms were listed that could result from either faults or contention. These symptoms are:

1) The reception of reconfiguration or status commands from a different FTP.

2) The disconnection of nodes that were previously connected to the IC network.

3) The loss of node responses or the occurrence of errors in the node responses.

In the following sections, the growth algorithm is examined with respect to each of these error symptoms to illustrate some of the issues involved in the IC network growth.

5.1.6.1 Symptom 1: Reception of Commands from a Remote FTP

The reception of reconfiguration or status commands from a remote FTP can result if either a babbler retransmits old command frames or multiple FTPs contend for the growth. If this

error symptom is due to one or more babblers, then the algorithm will detect the babblers through the execution and re-execution of steps 2 and 3. The babblers will be detected, because they will disrupt the node responses during both executions (it is assumed that intermittent babblers do not exist). Accordingly, the algorithm will modify the reconfiguration chains, disconnect the babblers, and continue growing the network.

Alternatively, if this error symptom is due to network contention, then the algorithm will eventually detect the occurrence via step 1 or step 3. Specifically, as the contending FTPs continue their respective growths, the FTPs will begin to "disrupt" each other (modify the other FTPs' virtual path through the network), and the algorithm will detect the network contention.

Under certain improbable circumstances, the growth algorithm may initially confuse multiple FTP contention with multiple babblers and cause network components to be falsely failed. However, this situation is rectified in the final phase of the growth, because all previously failed network elements are re-tested.

5.1.6.2 Symptom 2: Connected Nodes Have Been Disconnected

Nodes previously added to the IC network can be disconnected because of hardware faults that arise during the growth or multiple FTP contention. The loss of previously connected nodes will be detected in step 1 or step 3. In this situation, the algorithm always assumes that network contention is the cause of the problem. However, as stated, the loss of these connections may be due to hardware failures that manifest themselves during the growth. If faults are the cause of the problem, they will be detected during a subsequent regrowth of the network (by a different FTP or this FTP after its back off period has elapsed). Specifically, these faults will be isolated during the execution and re-execution of step 2.

5.1.6.3 Symptom 3: Node Responses are Lost or Have Errors

The loss of node responses (or errors in the node responses) can be due to hardware faults or the occurrence of network contention. If this error symptom is the result of faults, then the algorithm will detect these faults through the execution and re-execution of step 2. They will be detected, because both executions will have errors. As a result, the algorithm will modify the reconfiguration chains, bypass the faults, and continue the growth.

Under certain improbable circumstances, the growth algorithm may initially confuse multiple FTP contention with multiple faults and cause network components to be falsely failed. However, this situation is rectified in the final phase of the growth, because all previously failed network elements are re-tested.

5.1.7 Determination of a Back Off Period

If an FTP that is growing the IC network detects that another FTP is also attempting the growth, it will back off. That is, the IC Network Manager on this site stops performing the growth and delays a predetermined period of time. This back off period is calculated using the following equation:

Delay = [((# of triplex FTPs) - 1) + ((FTP_ID - 1) * 2)] * (Worst Case Grow Time)

The Worst Case Grow Time is the primary factor in the back off delay, because it is the basic offset necessary to avoid network contention. The procedure to compute this parameter for a given network topology is described in Section 5.2.3.3. A typical value for this parameter for the AIPS engineering model 5-node IC network is 2.6 seconds. This value is derived from empirically measured network transaction time and the worst case number of transactions that are required to grow the network for the engineering model IC network topology. The ((# of triplex FTPs) - 1) component is used to create a non-zero offset that is based on the maximum number of FTPs that could contend for the growth. Further, the (FTP_ID - 1) component is utilized to prioritize the FTPs contending for the growth. Finally, the factor of 2 is required to allow enough time to grow the IC network when consecutive numbered sites contend and back off.

The back off delay is not designed to minimize the IC network growth time in the presence of network contention. The delay is designed to ensure that the IC network can be successfully grown even if two or more FTPs contend for growth of the network.

5.1.8 The IC Network Diagnostics

As stated earlier, after the IC network is grown and all FTPs are accessible, IC network diagnostic tests are performed. These tests are used to detect hardware faults in the spare network components, to verify that network elements were not falsely failed during growth, and to exercise the network nodes. The diagnostics are performed by the IC Network Manager and are completed in a centralized manner on a layer by layer basis.

To determine if faults exist in an IC layer's spare links and nodes, the virtual communication path is reconfigured to employ these components. After the IC growth is completed, the IC Network Manager is capable of communicating with each node in the network. To verify that a spare element is working, it is cycled into the active path (using a reconfiguration chain) and an attempt is made to again communicate to all network nodes (via a status chain). If these nodes can still be reached without communication errors, then the newly utilized component is deemed working. Alternatively, if transmission errors occur, then the previous virtual path is restored and the element under test is marked as failed. This process is continued until all spare network components are examined.

To verify that the components failed during the IC network growth were not falsely failed, the IC Network Manager attempts to utilize these elements. If a failed link is being re-tested, then the Manager cycles it into the active virtual path and attempts to communicate to the network nodes. If all nodes respond without errors, then the link is presumed to be working. Conversely, if errors are encountered, then the previous path is restored and the component is confirmed to be faulty.

The diagnostics tests performed to recheck a failed node differ slightly from those used to test a failed link. Similar to the link verification process, the failed node is tested by attempting to communicate to it. However, as illustrated in Figure 5-1, each node can typically be reached by multiple paths. The diagnostic tests select one possible route, use a reconfiguration chain to actuate the path, and process the response from the chain to determine if the node was reached successfully. If errors were observed, then this route to the node is considered faulty, and another is selected. If all paths to the node under test have been checked and are faulty, then the node is marked as failed. Nonetheless, if a path to the node is found that appears error-free, then the node is deemed working. After such a node is re-instated as working, then any links that are now spare are checked for faults.

These diagnostic tests are also performed to exercise and stress the network nodes. The IC network growth focuses on the construction of a virtual path. This growth process does not comprehensively check the network nodes, because of the possibility of network contention. Accordingly, these diagnostics are employed to verify that all nodes operate as expected. For example, these tests verify that the nodes do not transmit on disabled ports, do not respond to the wrong address, and are capable of quick reconfiguration.

5.1.9 Network Contention Examples

To summarize the discussion of the IC network growth algorithm, two network contention examples are presented. These examples involve four triplex FTPs and are described in the Sections 5.1.9.1 and 5.1.9.2.

5.1.9.1 Example 1: Four FTPS Contend - Three FTPS Back Off

Example 1 is illustrated in Figure 5-6. In this scenario, each FTP powers up and attempts to grow the IC network at a different time. FTP 1, FTP 2, and FTP 3 eventually realize that other FTPs are attempting to grow the IC network, and each site backs off (the back off period is calculated using the aforementioned equation). FTP 4 does not detect network contention and completes the growth of the IC network. Accordingly, FTP 4 assumes the System Manager and IC network FDIR responsibilities, and it broadcasts the "Network is In_Service" message. When back off time for FTP 1 expires, the site broadcasts a "Site is Accessible" message. The System Manager on FTP 4 responds to the message by broadcasting an acknowledgement. Eventually, the back off times for FTPs 2 and 3 expire, and a similar procedure is performed. Finally, all of the FTPs are accessible and

network diagnostics are performed. After the diagnostic procedure has been completed, the IC User Communications are enabled.

5.1.9.2 Example 2: Four FTPS Contend - Four FTPS Back Off

Example 2 is illustrated in Figure 5-7. Example 2 is identical to Example 1 except that all four FTPs detect contention and back off. Since FTP 1 has the shortest back off time, it is the first site to "wake up". After its back off period expires, FTP 1 broadcasts a "Site is Accessible" message. The IC Network Manager on FTP 1 waits for a predetermined time period (greater than the worst case growth time, T) for a response from System Manager. Since the System Manager has not been assigned, FTP 1 times out and re-attempts to grow the network. FTP 1 completes the growth uncontested, because the other FTPs are still backed off. FTP 1 assumes the System Manager and IC network FDIR responsibilities. The System Manager broadcasts the "Network is In_Service" message and subsequently acknowledges each "Site is Accessible" message (FTP 2 sends the "Site is Accessible" message twice, because it received the "Network is In_Service" broadcast - see Section 5.1.2). Finally, after all FTPs are accessible, the network diagnostics are performed, and the IC User Communications are enabled.

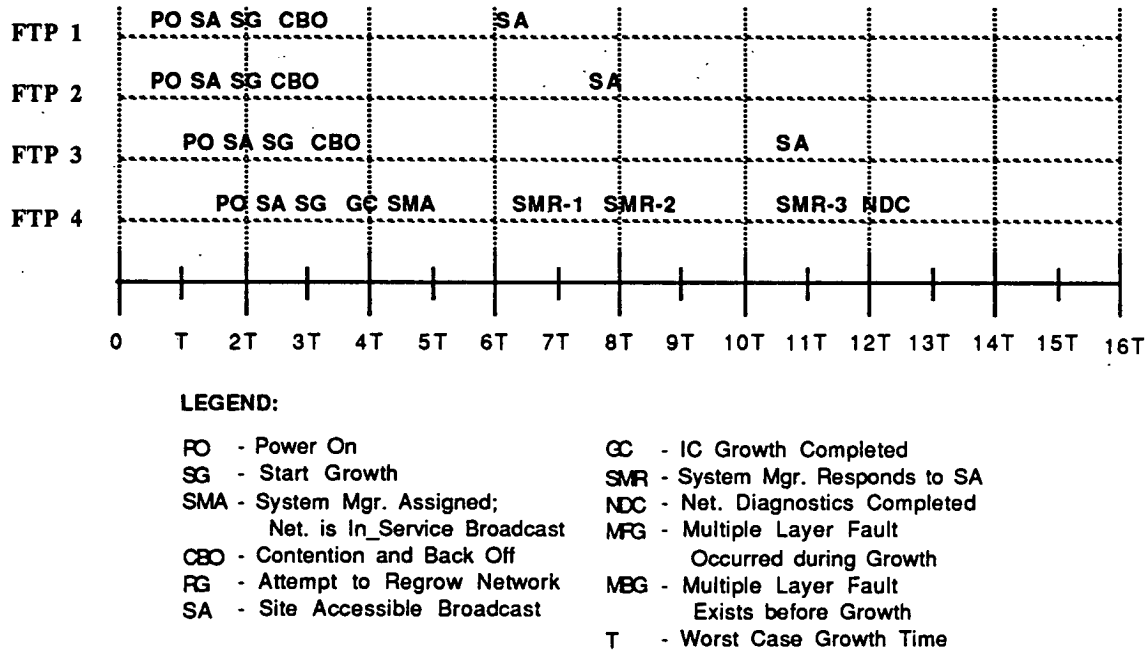


Figure 5-6. Network Contention Example - FTP 4 Grows the IC Network

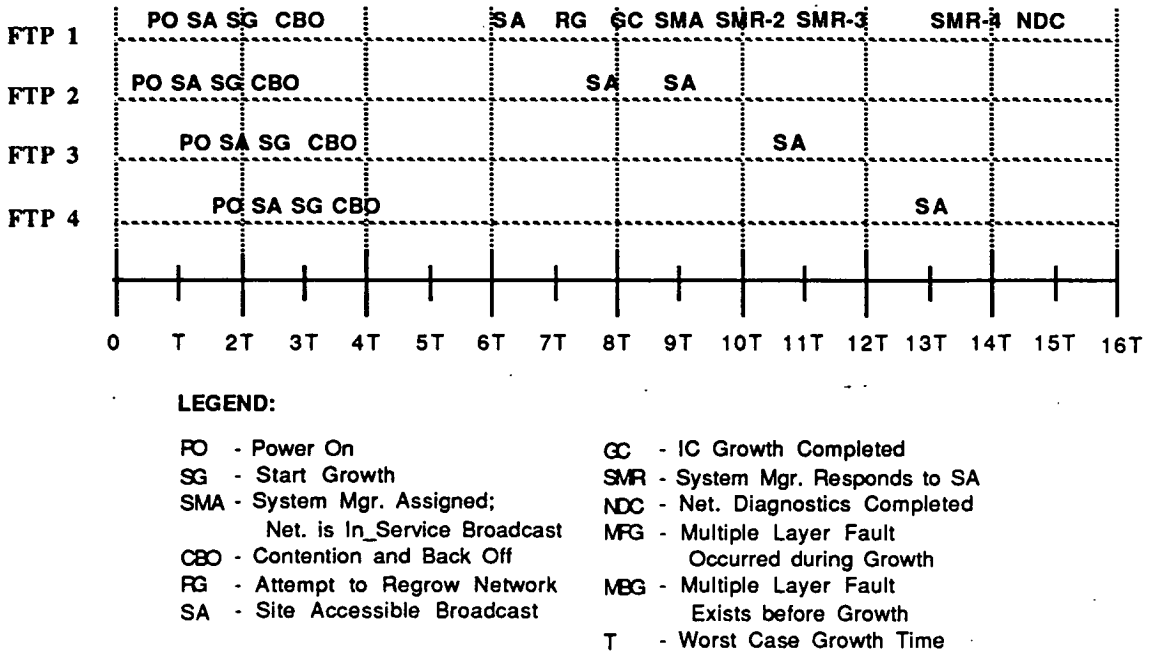


Figure 5-7. Network Contention Example - FTP 1 Grows the IC Network

5.2 IC Network FDIR Functional Requirements and Design

In contrast to the IC network growth, the network fault detection, isolation, and reconfiguration (FDIR) function is assigned to a particular FTP. The IC Network Manager that performs the IC network growth assumes the network FDIR responsibilities for all layers. That is, the three IC Layer Managers are assigned to the same FTP as the IC Network Manager that grows the IC network, and they are responsible for maintaining the layers of the IC network. These "IC Layer Manager to FDIR" assignments are changed only if a hardware fault necessitates it.

The functional requirements and design of the IC network FDIR process are discussed in this section.

5.2.1 The IC Network Layers and the IC Layer Managers

As illustrated in Figure 5-1, each layer of the AIPS IC network employs five nodes, seven inter-node links, and four FTP-node links. The IC network growth creates a virtual path through each layer to permit reliable inter-FTP communication. Given this topology, each layer has a spare node and three spare inter-node links. When necessary, these spares can be brought into service to reconfigure around faults or damaged components.

After the IC network is grown and the System Manager is activated, the IC network FDIR responsibilities are assigned. As outlined in Section 5.1.1, these responsibilities are given to the FTP that hosts the IC Network Manager which grows the IC network. This IC Network Manager allocates and initializes three IC Layer Managers. Each IC Layer Manager controls the FDIR activities for a particular layer.

The IC Network Managers that did not grow the IC network are considered Alternate IC Network Managers. Similarly, their corresponding IC Layer Managers are designated as Alternate IC Layer Managers. These processes are capable of assuming the IC network FDIR responsibilities but are dormant until such FDIR functions are migrated to them.

5.2.2 IC Network FDIR - Fault Detection

The presence of a fault (or damaged component) is detected during normal inter-computer communication. The fault (or faults) is perceived, because it disrupts the transmission from one site to another. More specifically, when the IC Communication Services on an FTP sends a message, three copies are transmitted. One copy is sent on each layer of the IC network. If a fault exists such that the virtual path on a layer is disrupted, then the copy transmitted on this layer will be corrupted. As a result, the destination FTP will correctly receive only two of the three copies. The destination FTP, specifically the ICIS Redundancy Management and Source Congruency process (discussed in Section ***), performs a suite of diagnostic tests to determine if this information loss was due to a fault in the network interface (the ICIS) or in the network layer. If the ICIS Redundancy Management procedure suspects that the communication problem is due to a layer fault, it notifies the the IC Network Manager (which may reside on a remote FTP) and identifies the questionable layer. Subsequently, the IC Network Manager takes the IC layer out of service, broadcasts this updated layer status to all FTPs, and sends a "repair" command to the corresponding IC Layer Manager. The IC Layer Manager inspects the layer to verify the presence of a fault, attempts to identify the fault, and reconfigures the layer around the failed component.

5.2.3 IC Network FDIR - Fault Analysis and Reconfiguration

When an IC Layer Manager receives a repair request, it executes a "verification" status chain to determine if all nodes in the questionable layer can be reached. If all nodes return an error-free response, then the Layer Manager presumes that the fault was transient or that it exists somewhere external to the layer. In either situation, the IC Layer Manager deems the layer to be working and accordingly notifies the IC Network Manager which informs the System Manager and all remote FTPs. Alternatively, if all nodes have errors, then it is presumed that the ICIS interface to the layer has failed. As a result, the IC Layer Manager informs the IC Network Manager and the FTP Resource Allocator that it is unable to perform FDIR on this layer. If the ICIS is working and the response from the verification status chain has one or more errors, then the layer under test has at least one faulty

component. The IC Layer Manager then invokes a fault analysis procedure to try to identify the type and location of the failed component. After this analysis is complete, the Layer Manager enters the fault reconfiguration phase and the faulty network element is bypassed. The next two sub-sections discuss the fault analysis and reconfiguration processes respectively.

5.2.3.1 Fault Analysis

The fault analysis process utilizes the results of the verification status chain to speculate on both the type of fault and its location within the IC layer. This hypothesis is constructed using two procedures: the node data analysis and the error analysis. (It should be noted that the assumption underlying all the deductive reasoning in these algorithms is that only one component has failed).

The data analysis process examines the response from the verification status chain to determine if a node is transmitting on a port that should be disabled. Initially, this algorithm decides if a babbler is present, because a babbler causes the network nodes to return invalid data. If such a fault is not detected, then this process analyzes the data from the error-free node responses, determines the nodes that have received messages, and examines the origin of these messages. If a non-failed, disabled port reports the reception of a message, the node adjacent to that port is transmitting on a disabled port (adjacent ports are always in the same configuration, either both enabled or both disabled). If a node is located with such a fault signature, then the fault is attributed to the node transmitting on the disabled port and the fault reconfiguration procedure is invoked. In contrast, if more than one node is found to have this fault, the analysis is deemed not successful, and the error analysis process is called.

The error analysis process attempts to deduce which component produced the set of errors present in the verification status chain. Although these errors may indicate several faulty network nodes, they usually result from only one faulty component. Of course not all sets of errors are capable of being analyzed. The input space of this procedure has many combinations which do not pinpoint a specific network component as being faulty. Further, as stated previously, the presumption underlying the error analysis is that only one component has failed.

If all nodes in the layer have errors, the error analysis algorithm attributes the errors to an interface node or interface link failure. If some nodes have errors and others do not, two possible failure modes are considered: (1) a failed link or node through which no transmission takes place or (2) a single node failure. The single node failure symptom could be indicative of a node which does not respond to commands but which continues to retransmit messages as it did before the failure. It could also be a node which itself is not failed but to whose address another node in the network responds. The single node failure is easy to diagnose since exactly one node in the status collection chain shows an error.

If two or more but fewer than all nodes have errors, then the remaining problem is to determine if the cause of those errors is a link or node whose transmission/retransmission function is no longer operational. The basic idea is that when a link or a node fails in this manner, then all nodes downline of this fault also have errors. The signature of such a failure is that these nodes form a treelike pattern in the network. It should be noted that another failure mode that produces a similar pattern of errors is a node which babbles on all outboard ports (outboard ports are all ports except the one through which the connection to the IC Layer Manager is made). To determine whether the observed errors fit the pattern for a failed link, node, or outbound babbler is a three step process. The first step is to identify a node which qualifies as the root of the failed tree. Such a node is one that had errors itself but that has an inboard port (the port which receives the commands sent by the IC Layer Manager) adjacent to a non-failed node. To prove this hypothesis, exactly one node should have this characteristic. If two or more such nodes exist, the fault is considered undiagnosable. Nonetheless, if a root is found, the second step is invoked to determine whether or not all nodes downline of the root had errors attributed to them. This is accomplished by a recursive algorithm. First, the algorithm processes information about the current node. The initial current node is the root of the failed tree. Next, the recursion procedure examines the nodes that are adjacent to the outboard ports of the current node (i.e. downline of the current node). If none of the adjacent nodes have errors attributed to them, then the desired pattern is not present and the fault is considered undiagnosable. However, if the nodes downline of the current node do have errors, then the recursion continues until every downline node has been visited. If a treelike pattern is established,the third step of the pattern checking process can proceed. This step verifies that all the nodes that have errors appear in the failed tree, i.e. no nodes with errors lie outside the tree. If nodes with errors are found outside the tree, the fault is considered undiagnosable. Alternatively, if all of these nodes are in the tree, the fault reconfiguration algorithm is called to make the final determination of whether or not the fault is due to a failed link, a failed node, or an outbound babbler. If the fault was diagnosable, the error analysis process informs the reconfiguration algorithm of its speculated type and location.

Figure 5-8 shows a network which has a broken link. In this situation, the verification status chain would not return responses from the shaded nodes in the figure. Since Node 2 is the only node with an inboard port facing a non-failed node, it is identified as the root of the failed tree. Furthermore, all nodes downline of Node 2 are failed and no nodes outside the tree had errors. Thus, error analysis identifies this fault as a failed link between Node 1 and Node 2 or a failure of Node 2. The final identification of the fault takes place during network reconfiguration.

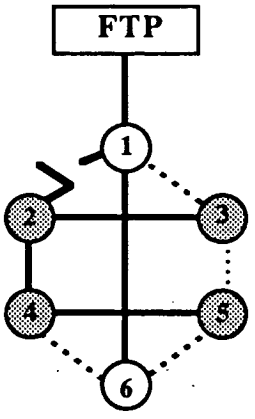


Figure 5-8. Identifying A Failed Link

5.2.3.2 Reconfiguration

The purpose of this process is to reconfigure the layer so as to restore error-free communication to all reachable, non-failed nodes. The reconfiguration action depends on the type of failure determined by the fault analysis process. The fault identified in this report is actually a hypothesis about what is causing the errors on the layer. The reconfiguration process, in effect, tests this hypothesis and then verifies that the layer is again fully operational. Therefore, the layer may go through several intermediate configurations before the reconfiguration process is complete.

The fault analysis process identifies five classes of faults: a babbler, a link or node failure, a node which transmits on a disabled port, a single node failure, and an undiagnosable failure. A separate strategy exists to deal with each of these fault classes.

The reconfiguration process is considered complete when the node status chain is executed on the reconfigured layer and does not detect any errors. The backup stratagem for dealing with unanticipated error phenomena which occur during a reconfiguration attempt is the layer regrowth (detailed in Section 5.2.3.3). This is also the strategy when the fault analysis is unable to diagnose the failure mode.

In general, reconfiguration strategies are designed to deal with both active and passive faults in the hardware. Passive faults are characterized by the non-retransmission of data, i.e. a barrier or obstacle to the flow of data in the layer. A disconnected cable is an example of such a fault; data cannot be retransmitted over this cable but transmission between other connections in the layer is not affected. Active faults are characterized by the disruption of data flow in the layer beyond the boundaries of the failed component itself. An ICIS with a transmitter stuck on high is an example of this type of fault; the stuck on condition is retransmitted throughout the layer, possibly disrupting transmissions between all layer inter-connections. Since different faults can produce identical error symptoms (e.g. a

broken root link and an ICIS transmitter stuck on high result in zero byte counts for all node responses), the reconfiguration algorithm must identify the specific cause of the problem so as to effect a repair.

When a babbler is detected in the layer, the layer is regrown. A babbler is an active fault that is detected by the ICIS at its receiving interface to the layer. (The ICIS cannot observe a stuck on high condition on its transmitting interface.) For a layer of N nodes which has a babbler, the cost of regrowing the layer is N + P chains, where P is the number of spare ports on the babbling node which must be tried. Strategies to lessen this cost are possible in layers that are either maximally branching or fully linear. Nonetheless, the current design uses regrowth to reconfigure the layer in which a babbler is present.

A failed node generates the same error pattern as a failed link. Thus, when the fault analysis reveals the presence of this failure mode, the reconfiguration algorithm must determine which fault has actually occurred and reconfigure the layer accordingly. It is first assumed that a link has failed. The failed link is disconnected, and an attempt to reach the failed node, i.e. the node immediately downline from the link, is made by using any spare ports on that node which are adjacent to non-failed nodes. The chain used to reconnect this node to the rest of the layer contains three transactions. The first two transactions enable the ports on either side of the new inboard link; the third transaction disables the former inboard port of this node in case the node adjacent to that inboard port is a babbler. If this strategy fails to restore communication with the failed node (possibly because no spare ports are available), data is assembled which will allow each branch of the failed tree to be reconnected to the active layer. This data consists of a list of nodes for each branch stemming from the failed node (i.e. a separate list for each set of nodes which lie downline of each of its outboard ports). Only one successful connection to any spare port on a branch is necessary to restore communication to the entire branch (and possibly to the failed node and all other nodes in the failed tree). Again a three transaction chain is used, this time for a different purpose. The first two transactions enable the ports on either side of the new link while the third transaction attempts to obtain status from the failed node. If the failed node correctly returns its status, the repair is complete and the absence of errors is verified by collecting status from every node in the layer. If the failed node is still not reachable, the port connecting this node to the present branch is disconnected and the proper functioning of the newly enabled link is verified. Then all nodes on this branch are removed from the failed node set. The net effect of this process is to restore communication with all reachable nodes in the layer while isolating the failed node. As communication to each branch is restored, the possible pool of spare links increases. Thus if any branch was not connected because of a lack of spare links, this branch is retried whenever a connection to another branch is successful. Any nodes which are still unreachable at the end of this exhaustive process are assigned a status of failed.

If a node retransmits valid data on a port which should be disabled, the node must be removed from the layer. This failure mode is distinguished from a babbler which is always

transmitting a random bit stream or is stuck on one. When a babbling port is identified, the adjacent port of the neighboring node is disabled. This neighboring node will not retransmit on its other enabled ports anything received from the disabled port. Furthermore, the node will ignore any random bit patterns it receives. However, if the neighboring node receives a request for its status on a disabled port (as might occur if a failed node is transmitting on a disabled port), it will transmit its status on all of its enabled ports. If this failed node is not removed, each time the Layer Manager asks for status from the node adjacent to this port, it would receive two valid commands to report its status, which will result in two status responses. However, only one response is expected. Once the first response is received by the Layer Manager, another node will be commanded to report its status. The second response of the node may interfere with the reply of a node whose transaction is later in the chain, making it appear that this next node has failed to respond correctly to a command. Once the failed node has been removed from the layer, status is collected from the remaining nodes to verify that in fact the fault has been identified and isolated. If errors are still detected in the layer, a full regrowth, with a complete set of diagnostic tests, is performed.

Removing a node is a simple matter if the node is a leaf; only the link connecting it to the layer needs to be disconnected. This is accomplished with one reconfiguration chain. If the failed node is not a leaf, the nodes downline from it need to be reconnected to the layer through alternate links. These downline nodes are added to a reconnection queue. Each of these nodes is also added to a set of unreachable nodes. The link connecting the inboard port of the failed node to the layer is then disabled. Next, an attempt is made to re-establish a connection to each isolated branch via a spare link from a node which is still reachable (i.e. is not a member of the unreachable node set). Only one such connection needs to be made to restore communication to all the nodes in the branch. After the new connection is enabled, the link connecting the failed node to this branch is disconnected. As each branch is reconnected, the nodes in that branch are removed from the reconnection queue. If any branch is successfully reconnected, the branches that were not connected during earlier attempts are tried again (since more spare links become available as communication is restored to nodes in other branches). This algorithm, while isolating the failed node, restores communication to every reachable node in the layer. Nodes which cannot be reached because earlier failures have depleted the pool of spare links are marked failed.

Figure 5-9 illustrates the steps needed to isolate a node from the layer. Suppose that Node 2 is to be removed from the layer. First the link connecting Node 1 to Node 2 is disabled. When this step is completed, Nodes 2, 3, 4, 5, and 6 are also isolated from the FTP as shown in part II. Node 2 is the root of a tree with two branches, each of which must be reconnected in turn. By enabling the link between Nodes 1 and 6 and disconnecting the link between Nodes 2 and 4, one of these branches is reconnected to the active layer as shown in part III. Finally, a link is enabled between Nodes 5 and 6 and the link between Nodes 2 and 3 is disabled. In this reconfiguration, Node 2 is isolated while preserving several links in the layer.

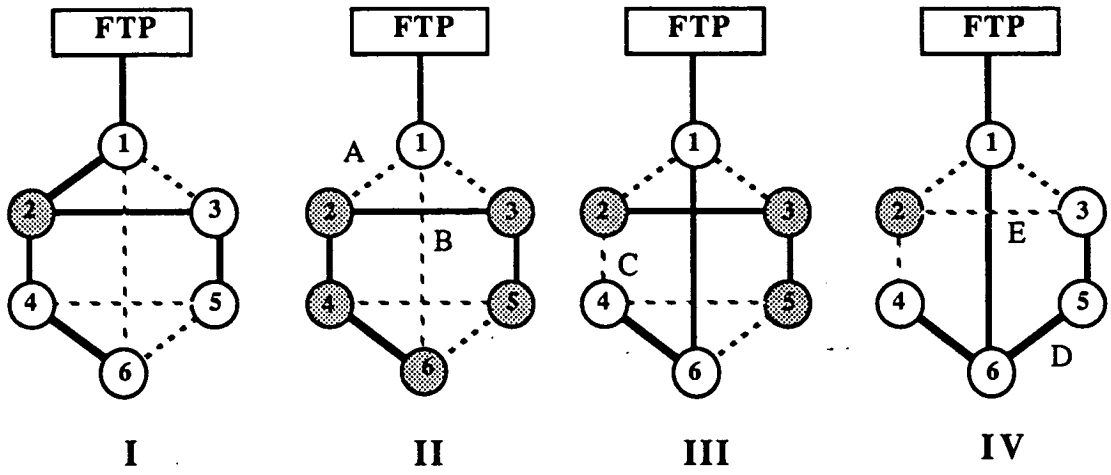


Figure 5-9. Removing A Node And Reconnecting Its Branches

A single node failure can occur: (1) if the failed node is a leaf node, (2) if its retransmission function still works correctly but its status reporting capability is impaired, or (3) if another node is responding to this node's address thus making this node appear failed. The failed node is isolated from the layer, as described in the previous discussion. However, care is taken not to address this node directly because of the possible addressing problem (the aforementioned fault scenario 3). After the node is isolated (disconnected from the active layer path), it is again queried for its status. If a valid response is received indicating the presence of a node which responds to the addresses of other nodes, the layer is regrown to isolate this faulty node. Otherwise, an attempt is made to find an alternate route to this node using any port except its previously failed inboard port. In addition to establishing a different route to the node, the reconfiguration command sent to this node disables the failed inboard port.

If the attempt to reconfigure the layer does not succeed in eliminating errors, then the IC Layer Manager regrows the layer (discussed in the following section). This is the back up reconfiguration strategy, used when all else fails.

5.2.3.3 IC Layer Growth

Layer growth is the process whereby the links between the nodes in the layer are enabled to form a virtual bus which supports communication among the layer subscribers (FTPs). Data flow in the layer is controlled by the configuration of the ports in each node. For a link to carry data between two nodes, the ports at either end of the link must both be enabled. Nodes retransmit messages received from an enabled port on its other enabled ports, but not on the port on which received the message. (The purpose of the retransmission is to maintain the integrity of the waveform and it only imposes a delay of one half the transmission clock period.) When a node receives a message addressed to

itself from any port, disabled or enabled, it carries out the command encoded in the message and then transmits its status on all of its enabled ports, including the port which received the message if that port is enabled. A node obeys reconfiguration commands sent by the IC Layer Manager by enabling or disabling its ports in accordance with the value of the port enable field in the command. Once the new configuration is in effect, the node returns a status message. For proper operation, there can be no loops in the layer. A data bit travels through each enabled link exactly once. Further, once it is grown, a layer operates like a time division multiplex bus.

Nodes are added one by one to the virtual bus. To determine which node to add next, the IC Layer Manager refers to the Layer Topology, a database which describes all physical interconnections that exist in the layer on a node by node basis. The algorithm used to add these nodes grows the bus in a treelike manner. Because of its resemblance to a tree, the nodes that are part of the virtual bus are said to be part of the active tree. The growth algorithm generates a maximally branching, minimum length path to every node in the layer. This configuration is later changed in order to repair faults. In addition to joining layer nodes into a virtual bus, the growth process is also concerned with enabling the communication paths to layer subscribers, the remote FTPs. This is accomplished by enabling the node ports adjacent to these devices and determining whether or not these components correctly obey the established communication protocols. The detection of protocol violations results in the subscriber being disconnected. In fact, the detection of a protocol violation when any new link is called into service results in the disabling of that link. Furthermore, the growth algorithm may employ a set of diagnostic tests that detect the presence of some malicious failure modes (such as nodes which transmit on disabled ports or nodes that respond to commands addressed to other nodes).

The layer growth algorithm assumes that, although hardware faults may be present in the layer before the growth process commences, no additional faults will occur while growth is taking place. However, if errors are detected during growth which indicate an additional failure, then the growth process is restarted. If a fault occurs repeatedly after a layer is partially grown, an intermittent failure can be inferred. Strategies to deal with short lived, intermittent failures need to be developed (beyond the scope of this functional design).

Layer growth begins by establishing an active link to the root node and ensuring that the root node has an outboard port to the rest of the nodes in the layer. Next, the remaining nodes are added to the active tree. Any nodes that are not connected to the active tree after this stage is complete are unreachable. After the layer is established through the active root link, the layer subscribers, that is the remote FTPs, are connected to the layer. Finally, status is collected from all nodes in the layer to verify that no failures have occurred in the layer during the growth process. Figure 5-10 summarizes the major steps in the layer growth algorithm. The following discussion details the logic employed in each major step.

For the growth of a layer to be considered successful, an active root link must connect the FTP to the layer. This implies the existence of a properly functioning ICIS and a root node that is able to communicate with the ICIS and at least one adjacent node. Establishing this connection is a two step procedure. In the first step, the hardware is put in a state which supports communication between the FTP and the root node. In the second step, the correct operation of this hardware is verified.

The first step in setting up a root link is to configure the root node so that the port adjacent to the ICIS is enabled and all of its other ports are disabled. The second step is to verify that the hardware involved in this root link is operating properly and that the root node can be used as a springboard to the rest of the layer. The absence of communication errors in the reconfiguration chain and the corresponding node response is evidence of a properly functioning communication link between the ICIS and the root node. If communication errors do not occur, then a determination is made about the root node's ability to function as a jumping off point for the addition of the remaining nodes in the layer. This decision is made by finding a link to an adjacent node which can be enabled without errors.

The algorithm for adding nodes to the layer is designed to conduct an exhaustive search for a properly functioning connection to every node in the layer. The failure of a single port of a node does not cause the entire node to be considered failed. However, some nodes may not be reachable by any path. The identity of these unreachable nodes is apparent only after this phase of the growth process is complete.

Repeat until growth is successful or two attempts fail to produce a stable layer:
    Establish a working connection to a root node.
    If an active root link is established then
        Add remaining nodes to the layer.
        Mark idle nodes failed.
        Add Remote FTPs.
        Collect Node Status from all layer nodes as defined by topology.
        Validate Layer Status.
        If no discrepancies in Layer Status then
            Layer is grown successfully.

Figure 5-10. The Layer Growth Algorithm

This stage of layer growth begins after a root link has been established. The root node becomes the first entry in a spawning queue, which is a data structure used to control the growth of the layer. An entry in this queue consists simply of a node that has been successfully added to the layer but from which growth has not yet taken place. Two positions are marked in the queue: the top and the next entry. The top holds the node in the queue from which growth is currently taking place. This node is called the spawning node.

The next entry is the next empty position in the queue. As nodes are added to the layer, they are placed on the spawning queue at the next entry point and the next entry point is advanced to an empty position in the queue. As growth of the layer proceeds, the topmost node in the spawning queue is removed from the queue and used as the jumping off point, or spawning node, for further growth. Each node in the spawning queue is processed in turn until the queue is empty.

The spawning node is processed on a port by port basis. The action taken depends on the type of element that is adjacent to each port. If the adjacent element is a remote FTP, the spawning node and the port that faces the FTP are recorded for future reference. Such ports are enabled after the growth of the layer is complete. However, if the adjacent element is a node whose status is idle, i.e. not yet part of the active tree, an attempt is made to enable the link to that node (which is referred to as the target node). If the attempt to enable the link between these nodes is not successful, the link is disconnected. Alternatively, if the attempt is successful, the target node is placed at the end of the spawning queue. When all ports of the spawning node have been processed, the next node in the spawning queue is removed and it becomes the new spawning node.

Figure 5-11 shows the entries made to the spawning queue for the growth of a fault-free, six node layer. Node 1, the root node, is the first entry. The three nodes adjacent to Node 1 are each added in turn to the layer. As each node is connected to the virtual path, it is added to the spawning queue. When all nodes adjacent to Node 1 have been added to the layer, Node 2 becomes the spawning node. Node 2 has an active link, an idle link adjacent to Node 3, and an idle link adjacent to Node 4. Since Node 3 is already active, the only node to be added to the layer from Node 2 is Node 4. The next spawning node is Node 6. Node 5, the only idle node adjacent to Node 6, is the last node added to the layer. Nodes 3, 4 and 5 each become a spawning node. However, since none of these nodes is adjacent to an idle node, no further nodes are added to the layer or to the spawning queue which is now empty.

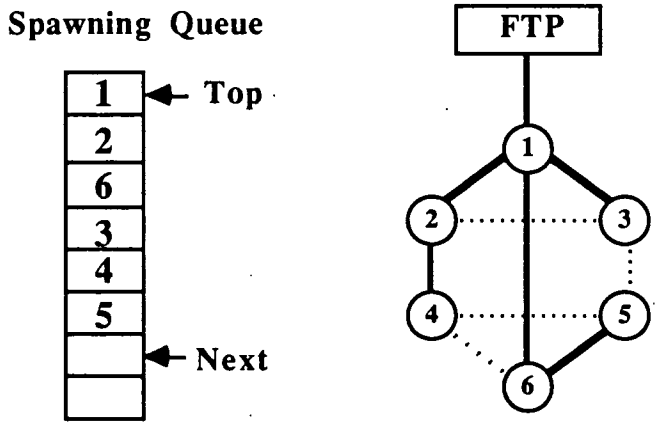


Figure 5-11. No Fault Growth Algorithm

The growth algorithm also detects and isolates babbling layer components, thus making it a useful backup strategem for layer maintenance. When a port of a spawning node adjacent to a babbling port is enabled, the babbler is detected because its transmissions interfere with the response of the spawning node. Following the detection of the babbler, the spawning node is sent a reconfiguration transaction instructing it to disable the port adjacent to the babbler, thus isolating the babbler from the rest of the properly functioning layer. The method works, because the layer links are full duplex (separate physical data links exist for the transmission and reception of data). As a result, the reconfiguration command reaches the spawning node through a path not corrupted by the babbler.

The use of the growth algorithm to isolate a node that is babbling on all of its ports is illustrated in Figure 5-12. Node 2 is shaded to denote it as the babbler. When Node 1 is the spawning node, the attempt to connect Node 2 fails, because the babbler violates the established communication protocols. Hence, Node 2 is not added to the spawning queue. Nevertheless, Nodes 6 and 3 are added as before. Node 6 is the second spawning node from which Nodes 4 and 5 are added to the active tree. When Node 3 becomes the spawning node, a second attempt is made to reach Node 2. This attempt is made, because a node may be babbling on one port only. When this attempt fails, Node 4 becomes the spawning node. Since Node 2 is still not in the active tree, a third and final attempt to reach Node 2 is made from Node 4. Although Node 2 is babbling, the ports facing it on Nodes 1, 3, and 4 are disabled and therefore its faulty transmissions cannot disturb other layer communication.

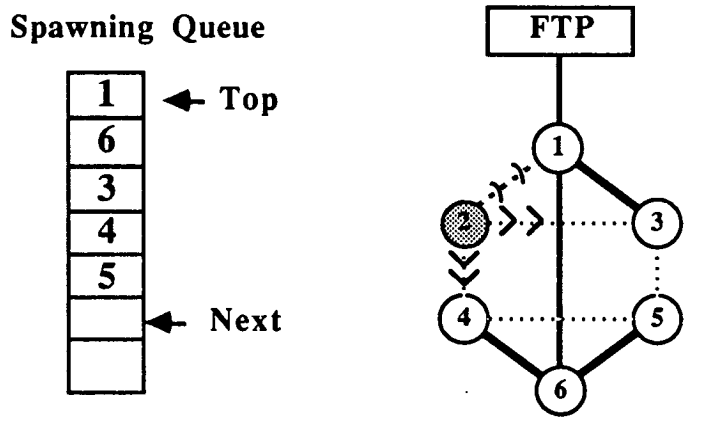


Figure 5-12. Layer Growth Used To Isolate A Babbling Node

As each node is added to the layer, a series of fault detection diagnostic tests *may* be performed. The tests are sequential in nature, and if any test fails, the remaining tests in the sequence are not performed. If this test sequence is employed throughout the layer growth, every layer link will be exercised.

The first diagnostic test determines if the link between two nodes can be activated. It is performed by enabling the link between the newly added node and its adjacent node. If the attempt to enable the link is successful, the link is left in the enabled state so that the next test can be executed. If the link is not enabled, the ports on either end of the link are failed.

The second test determines whether or not the adjacent node transmits on a port after it has been disabled. In this test, a configuration command is sent to the adjacent node over the newly enabled link instructing it to disable all of its ports. The node protocol is such that it carries out this command before transmitting a reply. A properly functioning node transmits a reply on all enabled ports to every command it receives. Since no ports are enabled, this message should not be transmitted. Thus, the node passes this test if a reply to the command is not received. If a node reply is received, then it is considered failed and its status is marked accordingly. Because the execution of this test, the adjacent node has all of its ports disabled prior to starting the third test.

The third test determines whether or not the newly added node retransmits a message on a disabled port. This test requires three transactions to be transmitted on the layer. The first transaction is sent to the newly added node commanding it to disable all of its ports except its inboard port (the port that connects the node to the established layer). The second transaction is sent to the adjacent node commanding it to enable the port facing the newly added node for one transmission only. The third transaction is sent to the newly added node asking for its status. If the newly added node is functioning properly, it will not retransmit any messages, including the command making up the second transaction, to the adjacent node. On the other hand, if it has failed such that it does retransmit a message on a disabled port, the adjacent node will send a reply which may or may not be transmitted back to the ICIS. In either case, the transmission of this message causes the valid message detector for the port facing the adjacent node to record the transmission and to return this information as part of its status message. The newly added node passes this third test if no message from the adjacent node is received and the status indicator for the port in question shows no valid message received on that port. However, if it fails the test, the status of the node is marked failed.

When the above three tests have been performed for every idle port of the newly added node, the newly added node remains configured such that only its inboard port is enabled. It is then ready for the last test.

If the preceding tests are completed without error, the fourth test is performed. This final test determines if the newly added node responds to commands sent to other nodes in the layer. In this test, each node in the layer is commanded to report its status, whether or not it is in the active tree. If an unconnected node responds to this command, it implies that the most recently connected node is responding to this address. Because of this protocol violation, this node must be disconnected from the active tree. Furthermore, its status is marked failed, since the address decoding function of the node is faulty. It is also possible that a previously connected node could respond with errors. This means that either this

node has recently failed or the most recently added node is talking out of turn. This last added node is then removed from the layer as described above. The node (or nodes) which had errors on the previous test are again queried for status. If the error indicators are gone, it confirms the talker out of turn hypothesis, and the status of the removed node is set to failed. If the errors are still present, it indicates that a failure has occurred during the growth process. In the former case, the growth process is continued. In the latter case, the growth process is restarted.

After all non-failed nodes are added to the virtual layer path, the ports adjacent to remote FTPs are enabled. An FTP which is facing a disabled port will not detect any layer activity. Accordingly, it may be attempting to use the layer at the time that the port is enabled. This could result in errors being detected in the node's reply to its configuration command. Thus any errors in the node status, which is returned after enabling the port towards a remote FTP, are ignored.

After the initial growth of the layer is completed, the status collection chain is executed through the root link. The chain response is analyzed to detect any discrepancies between the node status perceived by the IC Layer Manager and the status returned by this chain. This examination is performed to confirm or disprove the assumption that failures did not occur during the growth. If the data returned by this chain indicates the presence of a babbler or failed nodes which the IC Layer Manager reported as active, then a discrepancy exists between the actual state of the layer and its state as decided during layer growth. It cannot be determined whether these failures occurred during or after the growth of the layer. Thus, the layer is regrown. If this second try is unsuccessful, an intermittent failure exists on the layer. The present algorithm does not handle intermittent faults. Hence, the layer is declared to be inactive.

5.2.4 IC Network FDIR - Layer Status Notification

After the IC Layer Manager has reconfigured the IC layer, the IC Network Manager and System Manager are notified. If the layer has been repaired successfully, its status is marked to in_service. Alternatively, if the layer could not be restored, it is kept out_of_service. In either case, the IC Network Manager broadcasts a messages to all FTPs informing them of the updated layer status.

5.3 IC Network Growth Software Specifications

5.3.1

Process Name: IC Network Manager

Inputs:
- Layer FDIR Request
- Layer Identifier
- Layer Topology
- Layer Usability

	Layer Status
Outputs:	Layer Status
Requirements Reference:	IC Network Growth Functional Requirements, Section 5.1
Notes:	None.

Description:

Each triplex or higher redundancy FTP may create an instance of the IC Network Manager process to attempt to grow the IC Network. Since multiple FTPs may attempt the IC network growth but only one completes it, only one FTP that activates this process will grow the network. If an FTP does not perform the IC network growth, the associated instance of the Network Manager is deactivated and the process becomes an Alternate IC Network Manager. Such a Manager is only re-actuated if a network FDIR function is migrated to the FTP that hosts it.

Conversely, if this IC Network Manager does grow the IC network, it allocates and initializes three IC Layer Managers. Each Layer Manager is assigned to a particular layer and is responsible for the corresponding layer FDIR. After the Layer Managers are activated and initialized, the IC Network Manager suspends itself.

The arrival of a network FDIR request causes the IC Network Manager to resume. The request identifies a layer that is speculated to have a faulty component. The IC Network Manager takes the IC layer out of service, broadcasts this updated layer status to all FTPs, and sends a *repair* command to the corresponding IC Layer Manager.

When the IC Layer Manager completes the FDIR process, it notifies the IC Network Manager process by setting the Layer Usability field to *repaired* and informs it of the layer status. The Network Manager process subsequently informs the System Manager and broadcasts the updated layer status to all FTPs. After the layer status message has been sent, the IC Network Manager suspends until another IC network FDIR request is posted.

5.3.2

Process Name:	Send Site is Accessible Message
Inputs:	FTP Identifier
Outputs:	Message to System Manager
Requirements Reference:	IC Network Growth Functional Requirements, Section 5.1.2
Notes:	None.

Description:

The Send Site is Accessible Message process is part of the IC network initialization protocol. It is invoked by either the FTP Resource Allocator or the IC Network Manager to inform the System Manager that this FTP has completed its local initialization procedure. The process uses the FTP ID as an input parameter and employs the IC Communication Services to broadcast the Site is Accessible message.

5.3.3

Process Name:	Send Network is In_Service Message
Inputs:	FTP Identifier
Outputs:	Network is In_Service Broadcast
Requirements Reference:	IC Network Growth Functional Requirements, Section 5.1.2, 5.1.3
Notes:	None.

Description:

The Send Network is In_Service Message process is part of the IC network initialization protocol. It is invoked by the System Manager to inform *all* activated FTP Resource Allocators that the IC network is in_service and that the System Manager has been assigned to an FTP. The process uses the FTP ID as an input parameter and employs the IC Communication Services to broadcast the Network is In_Service message.

5.3.4

Process Name:	Acknowledge Site is Accessible Message
Inputs:	Site is Accessible Message Initialized FTPs Redundancy of Initialized FTPs Location of System Manager System Status
Outputs:	Site is Accessible Acknowledgement Initialized FTPs Redundancy of Initialized FTPs Location of System Manager System Status

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.2

Notes: None.

Description:

The Acknowledge Site is Accessible Message process is part of the IC network initialization protocol. It is invoked by the System Manager to inform *all* activated FTP Resource Allocators that the IC network is in_service and that the FTP which sent the Site is Accessible message has completed its local initialization. This acknowledgement is composed of: the FTPs that have completed initialization, the redundancy levels of these initialized FTPs, the location of the System Manager, and other status information.

The process employs the IC Communication Services to broadcast the Site is Accessible acknowledgement.

5.3.5

Process Name: Perform Network Diagnostics

Inputs:
IC Network Topology
IC Network Configuration
IC Network Status
Status Collection Report

Outputs:
IC Network Status
IC Network Configuration

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.8

Notes: None.

Description:

After the IC network is grown and all FTPs are accessible, the Perform Network Diagnostics process is called by the IC Network Manager. This procedure is used to detect hardware faults in the spare network components, to verify that network elements were not falsely failed during the growth, and to exercise the network nodes. It executes the diagnostic tests in a centralized manner on a layer by layer basis.

To determine if faults exist in an IC layer's spare links and nodes, the virtual communication path is reconfigured to employ these components. After the IC growth is

completed, the IC Network Manager is capable of communicating with each node in the network. To verify that a spare element is working, it is cycled into the active path (by executing a reconfiguration chain from the ICIS interface) and an attempt is made to again communicate to all network nodes (via the Layer Status Collection process as described in Section 5.4.7.1). If these nodes can still be reached without communication errors, then the newly utilized component is deemed working. Alternatively, if transmission errors occur, then the previous virtual path is restored and the element under test is marked as failed. This process is continued until all spare network components are examined.

To verify that the components failed during the IC network growth were not falsely failed, the IC Network Manager attempts to utilize these elements. If a failed link is being re-tested, then the Manager cycles it into the active virtual path and attempts to communicate to the network nodes. If all nodes respond without errors (again using the Layer Status Collection procedure), then the link is presumed to be working. Conversely, if errors are encountered, then the previous path is restored and the component is confirmed to be faulty.

The diagnostic tests performed to recheck a failed node differ slightly from those used to test a failed link. Similar to the link verification process, the failed node is tested by attempting to communicate to it. However, each node can typically be reached by multiple paths. The diagnostic tests select one possible route, use a reconfiguration chain to actuate the path, and process the response from the chain to determine if the node was reached successfully. If errors were observed, then this route to the node is considered faulty, and another is selected. If all paths to the node under test have been checked and are faulty, then the node is marked as failed. However, if a path to the node is found that appears error-free, then the node is deemed working. After such a node is reinstated as working, then any links that are now spare are checked for faults.

These diagnostic tests are also performed to exercise and stress the network nodes. The IC network growth focuses on the construction of a virtual path. This growth process does not comprehensively check the network nodes, because of the possibility of network contention. Accordingly, these diagnostics are employed to verify that all nodes operate as expected. For example, these tests verify that the nodes do not transmit on disabled ports, do not respond to the wrong address, and are capable of quick reconfiguration.

As stated earlier, several of the network diagnostic tests utilize the Layer Status Collection procedure. The Collection algorithm returns a Status Collection Report. This Report represents the results of a preliminary error analysis that is conducted on the nodes' status response. This Report is reviewed by the Perform Network Diagnostics process to determine if errors occurred in a particular test.

5.3.6

Process Name:	IC Network Growth
Inputs:	IC Network Topology IC Network Configuration IC Network Status Iteration Results Iteration Completed Error Analysis Report
Outputs:	IC Network Status IC Network Configuration Network Grown Network Grown by This FTP
Requirements Reference:	IC Network Growth Functional Requirements, Section 5.1.5
Notes:	None.

Description:

The IC Network Growth process is responsible for executing the IC network growth algorithm. It repeatedly invokes the Execute Iteration function to incrementally add nodes to the IC network. After each growth iteration has completed, it calls the Iteration Error Analysis procedure to decide if the nodes were appended correctly. If communication errors occur in an attempt to add a set of nodes, the IC Network Growth process determines if contention occurred or hardware faults exist. Further, for each step of the growth, it decides which set of nodes to add next and by which routes to connect them.

First, this process enters an initialization phase. This phase involves setting: the status of the nodes and ports to idle (which initializes the IC Network Status database), each port of IC Network Configuration to disabled, the Network_Grown flag to false, and the Network_Grown_by_This_FTP flag to false.

As stated earlier, this process grows the IC Network incrementally. Each increment, it tries to append three target nodes to the established IC network path. It determines the next nodes to add, the path by which these nodes can be reached, and initializes the required reconfiguration chains and data in the ICIS Dual Ported Memory to establish this route. It then invokes the Execute Iteration process to attempt to create the path and waits for the iteration to complete.

The completion of each growth iteration is indicated by the Iteration Completed flag. When the flag is set, the IC Network Growth process calls the Iteration Error Analysis function to determine if errors disrupted the execution of reconfiguration or status collection chains. The Error Analysis process queries the ICIS registers and nodes' response data to detect the presence of a transmission error (or errors). This procedure returns its results in the Error Analysis Report record. The IC Network Growth function examines this record to decide the resultant course of action. If errors were not encountered, the Network Status and Network Configuration databases are updated to reflect the addition of the target nodes. Alternatively, if one or more errors occurred, this process follows the growth algorithm, as described in Section 5.1.5, to distinguish between network contention and hardware faults.

If an error pattern indicates that contention for the network growth occurred, this process calls the Back Off procedure, providing the FTP ID as an input parameter. On the other hand, if hardware faults on one or more layers are hypothesized, then it examines the Network Topology with respect to speculated faults and, if possible, devises an alternate route to the target nodes. Subsequently, the chains and data in the ICIS DPM are updated, and the Execute Iteration procedure is called again.

The growth of the IC network is composed of three general steps: (1) establish a root connection, (2) add the network nodes, and (3) enable the remote FTPs. Initially, the IC Network Growth algorithm attempts to create three root link connections to the IC network. The establishment of such connections requires that the root nodes be configured so that their ports which face the FTP (that is performing the IC network growth) are enabled. If one or more of these communication links is faulty, the FTP can not continue the growth and backs off. Conversely, if these links are established, then the process begins to add the nodes to the IC network.

The algorithm for determining the next set of nodes to append to the IC network path is similar to that employed for the IC layer growth. It uses the Network Topology, Network Status, and Network Configuration databases to decide which nodes can be added and the route to reach them. Furthermore, it utilizes the spawning queue methodology as detailed in Section 5.2.3.3.

Finally, after all of the reachable nodes have been connected to the IC network, the remote FTP subscribers are given access to the network. This is performed by enabling the node ports that face these sites.

If this process backs off, then when its back off period expires, it invokes the Send Site is Accessible Message process. The Site is Accessible process is called to determine if the IC network was grown while the FTP was suspended. If the network has been established, the IC Network Growth process sets the Network_Grown flag to true and returns to the calling procedure. If the IC network has not been constructed, it re-attempts to perform the growth.

If this process completes the growth of the IC Network, it notifies the FTP Resource Allocator and System Manager by setting the Network_Grown_by_This_FTP Flag.

5.3.7

Process Name: Execute Iteration

Inputs:
IC Network Topology
IC Network Configuration
IC Network Status
Chain to Add Target Nodes

Outputs:
IC Network Status
IC Network Configuration
Iteration Completed
Iteration Results

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.5

Notes: None.

Description:

The Execute Iteration process is called by the IC Network Growth procedure to add three target nodes to the established virtual communication path. This process attempts this addition by executing the three steps that comprise the iteration:

1. Request the status of the previously added nodes.
2. Execute a reconfiguration chain to append the three target nodes.
3. Request the status of the previously added nodes and the target nodes.

This is completed by executing three procedures: Execute Step 1 of the Iteration, Execute Step 2 of the Iteration, and Execute Step 3 of the Iteration.

This process correlates the error information returned from the aforementioned steps and uses the Iteration_Results record to send the information to the IC Network Growth procedure. In addition, the completion of an iteration is signaled by setting the Iteration_Completed flag.

5.3.7.1

Process Name: Execute Step 1 of the Iteration

Inputs: Node Status Collection Chain

Outputs: Chain Data Response

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.5

Notes: None.

Description:

The Execute Step 1 of the Iteration executes the Node Status Collection Chain which queries each node in the IC network (whether connected or not) for its status. The process saves the relevant ICIS register information and the nodes' status response (in the Chain Data Response record) to be subsequently examined. This information must be recorded after the execution of the chain, because the execution of steps 2 and 3 over-writes the return data.

5.3.7.2

Process Name: Execute Step 2 of the Iteration

Inputs: IC Network Reconfiguration Chain

Outputs: Chain Data Response

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.5

Notes: None.

Description:

The Execute Step 2 of the Iteration executes the IC Network Reconfiguration Chain which attempts to configure three target nodes so as to add them to the established virtual IC path. The process saves the relevant ICIS register information and the nodes' status response (in the Chain Data Response record) to be subsequently examined. This information must be recorded after the execution of the chain, because the execution of step 3 over-writes the return data.

5.3.7.3

Process Name: Execute Step 3 of the Iteration

Inputs: Node Status Collection Chain

Outputs: None

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.5

Notes: None.

Description:

The Execute Step 3 of the Iteration executes the Node Status Collection Chain which queries each node in the IC network (whether connected or not) for its status. The process *does not read* the relevant ICIS register information or the nodes' status. This data is read directly by the Iteration Error Analysis procedure.

5.3.8

Process Name: Iteration Error Analysis

Inputs:
IC Network Topology
IC Network Configuration
IC Network Status
Iteration Results

Outputs:
IC Network Status
IC Network Configuration
Error Analysis Report

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.5

Notes: None.

Description:

This Iteration Error Analysis process reviews the results of the three steps of the iteration to determine if errors occurred during their execution. The data from steps 1 and 2 are passed into this procedure while the data from step 3 is retrieved directly from the ICIS DPM. The results attained from this examination are returned to the IC Network Growth process in the Error Analysis Report.

The resultant Error Analysis Report provides the IC Network Growth Algorithm with a summary of the error information obtained from a preliminary analysis of the response data from the reconfiguration and status collection chains. When the ICIS transmits messages on the layer to a node, it observes aspects of the communication and records those observations in registers and buffers for later processing. This constitutes a first stage of fault detection, and it includes detection of: the failure of a node to respond to a command in a reasonable length of time, the presence of transmission errors on the layer, an incorrect number of bytes in a response, and other violations of the communication protocol. In addition to detecting errors on transactions to individual nodes, the overall performance of the layer is monitored for failures which impede the proper functioning of the contention sequence. These failures include a babbler which is flooding the bus with meaningless signals and a data line which is holding the layer in a "stuck on one" condition.

This Error Analysis Report presents a summary of the information provided by the ICIS with conclusions drawn about the following error conditions: an interface failure, a babbler, and individual errors detected for each node. If the summary reports that an interface failure has occurred, it also states whether the cause is a failed ICIS or a failed channel connected to the active ICIS. If the summary reports that a babbler is present on the layer, it also specifies whether the babbler was detected during contention for the layer or during data transmission. When either of these errors are present, no further data is provided since the integrity of this data is in question. Finally, if neither an interface failure or a babbler is detected, an error indicator is provided for each active node in the layer. This error indicator simply notes that an error has occurred. The error could be due to a variety of causes, including a no response error, an HDLC protocol violation, or a check sum error. The type of error is is not passed back in the Error Analysis Report since the IC Network Growth Algorithm does not require this level of granularity.

5.3.9

Process Name:	Re-execution Delay
Inputs:	FTP Identifier
Outputs:	None
Requirements Reference:	IC Network Growth Functional Requirements, Section 5.1.5
Notes:	None.

Description:

This Re-execution Delay process detains the re-execution of a Growth Iteration to avoid successive network contention incidences. Each FTP will delay a different non-zero length of time before re-executing steps 1 - 3. This delay is incorporated to ensure that if two or more FTPs "directly contend" during the first execution of the chains, then they will not contend during the second execution. (In this context, *direct contention* occurs when two or more chains are executed at exactly the same time by different FTPs, thus causing the loss of or errors in the expected node responses). The delay that is imposed must be greater than the time required to execute and process steps 1 - 3 (the Delay >= FTP_ID * worst case time to process the iteration).

5.3.10

Process Name: Back Off

Inputs: FTP Identifier
Worst Case Growth Time

Outputs: None

Requirements Reference: IC Network Growth Functional Requirements, Section 5.1.7

Notes: None.

Description:

If an FTP that is growing the IC network detects that another FTP is also attempting the growth, it will back off. That is, the IC Network Manager on this site stops performing the growth and delays a predetermined period of time. This back off period is calculated using the following equation:

Delay = [((# of triplex FTPs) - 1) + ((FTP_ID - 1) * 2)] * (Worst Case Grow Time)

The Worst Case Grow Time is the primary factor in the back off delay, because it is the basic offset necessary to avoid network contention. The ((# of triplex FTPs) - 1) component is used to create a non-zero offset that is based on the maximum number of FTPs that could contend for the growth. Further, the (FTP_ID - 1) component is utilized to prioritize the FTPs contending for the growth. Finally, the factor of 2 is required to allow enough time to grow the IC network when consecutive numbered sites contend and back off.

The back off equation is not designed to minimize the IC network growth time in the presence of network contention. The equation is designed to ensure that the IC network can be successfully grown even if two or more FTPs contend for growth of the network.

This Back Off process employs the aforementioned equation to determine the back off period and then suspends the IC Network Manager for this time period.

5.4 IC Network FDIR Software Specifications

5.4.1

Process Name: IC Layer Manager

Inputs:
Layer Identifier
Layer FDIR Request
Layer Topology
Layer Status
Layer Topology

Outputs:
Layer Status
Layer Usability
Layer Configuration

Requirements Reference: IC Network FDIR Functional Requirements, Section 5.2.1, 5.2.3

Notes: None.

Description:

An instance of this process is created for each layer of the IC network. Each process remains in a quiescent state until it is activated by the IC Network Manager. At any given time, only one actuated IC Layer Manager exists for a specific layer. The activation of this Layer Manager only requires the scheduling of the process on the associated FTP. Memory allocation, process instantiation, and the initialization of variables used by this process will already have taken place. As a result, the actuation of a Layer Manager can be accomplished very quickly if necessary.

When a Layer Manager process is created, some software initializations take place which need to be performed by this process only once. These include: obtaining the Layer Identifier of the layer it will manage, reading the Layer Topology from the Layer Topology Database, and retrieving a copy of the Layer Status. This sequence is accomplished during the power on phase of system operation. This preliminary work is not considered part of the routine operation of the Layer Manager.

Once it has been activated, the Layer Manager is responsible for maintaining its assigned layer. It uses the Layer Status Collection procedure (discussed in Section 5.4.7.1) to query the layer to determine: (1) if a fault exists and (2) if the interface from this FTP to the layer is working. If this query does not detect a fault, then the IC Layer Manager presumes that the fault was transient or lies external to the layer. Accordingly, the process informs the IC Network Manager and returns the layer status to *in_service*. Alternatively, if the FTP interface to this layer is faulty, it notifies the IC Network Manager and FTP Resource Allocator and assists in the migration of the Layer Manager function. If the interface is working and a layer fault exists, the IC Layer Manager invokes the error analysis and reconfiguration subprograms to perform layer FDIR (described in Sections 5.4.7).

Each time the Layer Manager initializes or reconfigures a layer in response to a call from the IC Network Manager, it indicates that it has completed its actions by writing a value of *repaired* to the Layer Usability field. In addition, it indicates the status of the layer in the Layer Status field, either *in_service* or *out_of_service*. The IC Layer Manager also informs the IC Network Manager of the existence of hardware faults and the occurrence of transient faults.

5.4.2

Process Name:	IC Layer Growth
Inputs:	Layer Identifier Layer Topology Diagnostics Option Layer Status Layer Configuration
Outputs:	Layer Status Layer Configuration
Requirements Reference:	IC Network FDIR Functional Requirements, Section 5.2.3.3
Notes:	None

Description:

This process makes two attempts to grow the designated layer which is specified in the IC Layer Topology Database. The Layer Topology describes all of the interconnections which exist in the layer on a node by node basis. Layer growth is accomplished by a set of nested subprograms. The outermost subprogram verifies and validates the results of a second inner subprogram which assumes that, although hardware faults may be present in the layer

before the growth process commences, no additional faults will occur while growth is being performed. This inner subprogram conducts the majority of the layer growth. It calls other subprograms to perform the layer growth and then returns a boolean parameter to its caller indicating whether or not the layer has been grown successfully. For growth of a layer to be considered successful, an active root link must connect the FTP to the layer, and all non-failed nodes in the layer must be part of the active tree. If the subprogram indicates that growth is not successful, it is called a second time. Alternatively, if the growth is successful, the outermost subprogram of the IC Layer Growth process executes a status collection chain. Further, it analyzes the resultant status data looking for any discrepancies between the nodes' status as perceived by the Growth procedure and that returned by the chain. This is to confirm or disprove the assumption made by the inner growth subprogram that failures did not occur in the layer during growth.

If the data from this status chain indicates the presence of a babbler, a failed ICIS, or failed nodes which the growth process reported as active, then a discrepancy exists between the real state of the layer and its state as recorded by the inner subprogram. It can not be determined whether these failures occurred after or during the layer growth. Thus, if a discrepancy exists, the layer is regrown. If the second try is unsuccessful, a serious problem exists on the layer requiring either a function migration or operator intervention to correct the problem. The choice of action is made by the System Manager.

As mentioned, faults can occur during layer growth. If such faults are detected, the growth process is restarted. This subprogram tries up to two times to grow the layer. If faults continue to occur during the growth process, then the subprogram informs the IC Layer Manager that growth was not successful.

The growth of the layer begins by establishing an active root link to the root node and ensuring that this root node has a port which can be used as the springboard to the rest of the nodes in the layer. If the root link is working, the remaining nodes are added to the active tree. This process conducts an exhaustive search for a properly functioning connection to every node in the layer. Once a particular connection is established, the status of the associated node is upgraded to active. The failure of a single port of a node does not cause the entire node to be considered failed. Some nodes may not be reachable by any path. However, the identity of these unreachable nodes will be apparent only after this phase of the growth process is complete. If a node has a status of idle after layer growth is completed, then it is not reachable by any port and its status is changed to failed. In addition, after the layer is established through the active root link, the ports adjacent to remote FTPs are enabled.

The growth process is summarized below. Further details on each aspect of the process are available in the indicated sections.

Repeat until growth is successful or two attempts fail to produce a stable layer
    Establish a working connection to a root node (5.4.3)
    If an active root link is established then
        Add remaining nodes to the layer (5.4.4)
        Mark idle nodes failed
        Add Remote FTPs (5.4.6)
        Collect Node Status from all layer nodes as defined by topology (5.4.7.1)
        Validate Layer Status
        If no discrepancies in Layer Status then
            Layer is grown successfully

5.4.3

Process Name:	Establish Root Link
Inputs:	Layer Identifier Layer Topology Diagnostics Option Layer Status Layer Configuration
Outputs:	Layer Status Layer Configuration Spawning Queue Active Root Link
Requirements Reference:	IC Network FDIR Functional Requirements, Section 5.2.3.3
Notes:	None

Description:

This process is the first step in the growth of a layer. Its job is to set up a properly functioning a connection to the layer. The hardware involved in the connection consists of an ICIS, the root node, and the link between them. Establishing the connection is a two step procedure. It requires that this hardware be configured to support communication between the FTP and the root node and that the operation of this hardware be verified.

If a successful connection to the root node can be made, the value of the Active Root Link flag is set to true to indicate to the calling subprogram that a communication link has been established. Additionally, the Spawning Queue is initialized with the root node. Conversely, if no root link is established, the value of the Active Root Link flag is set to false.

To set up a root link, the root node is configured so that its port facing the ICIS is enabled and all of its other ports are disabled. This is accomplished by preparing a reconfiguration command (causing it to configure its ports as described) and then executing a chain which sends this command to the root node. After the chain is executed, the response from the root node is examined to determine if any communication errors occurred.

The second step in setting up the root link is to verify that the communication hardware is operating properly and that this root node can be used as a springboard to the rest of the layer. The absence of errors in the first step is evidence of a properly functioning full duplex communication link, and it implies that the ICIS and node hardware is fully operational. If the Diagnostics Option has been selected, a full set of diagnostic tests are conducted on the root node. These are described in more detail in Section 5.4.5. If the root node passes all the diagnostic tests or if the tests are bypassed because the Diagnostics Option is not chosen, a determination is made about the ability of the root node to function as a jumping off point for the addition of the remaining nodes in the layer. If diagnostic tests are performed, this determination is made by identifying a non-failed port on the root node which is adjacent to another node. This is performed by querying the Layer Topology and the Node Status information. However, in the case when diagnostic testing is bypassed, this is accomplished by finding a link to an adjacent node which can be enabled to support full duplex communication. If such a link is found, the status of the root node, its port facing the ICIS, and the ICIS interface is marked active. Furthermore, the configuration of the root node is recorded in Layer Configuration. The Layer Configuration indicates that this node's enabled port is designated as Inboard and the other ports are specified as Idleport. This process is then complete.

The preceding paragraph describes the actions taken by this process if no protocol errors are detected when the configuration command is sent to the root node. When errors are detected, they are processed before a second try is made. The error processing proceeds as follows. If the error detected is a channel failure, a retry is not undertaken since it is unlikely that the channel can be restored in time to make this a viable root link. Instead, the Interface Status is marked Failed Channel and the Layer Manager informs the IC Network Manager. If any other errors are detected such as no response or HDLC protocol errors, the same configuration chain is simply run a second time. If errors are detected on the second try, the ICIS and the status of the port adjacent to the ICIS are marked failed, and the IC Layer Manager informs the IC Network Manager of the fault.

5.4.4

Process Name: Adding Nodes to Layer

Inputs:
Layer Identifier
Layer Topology
Diagnostics Option

	Layer Status Layer Configuration Spawning Queue
Outputs:	Layer Status Layer Configuration Layer Subscribers
Requirements Reference:	IC Network FDIR Functional Requirements, Section 5.2.3.3
Notes:	None

Description:

This growth algorithm generates the shortest path from the source FTP to any node in the layer. Furthermore, if a path exists to any node in a layer, this algorithm ensures that it will be found and activated, even if the layer is degraded by failures.

This subprogram is called into service after Establish Root Link (Section 5.4.3) has established a fully operational link to a root node of this layer. The root node is the first entry in the spawning queue, a data structure that is used to control the growth of the layer. An entry in the queue consists simply of the number of the node which has been successfully added to the layer but from which growth has not yet taken place. Two pointers are used to mark positions in the queue: the Top and the Next Entry. The Top points to the node in the queue from which growth is currently taking place. This node is called the spawning node. The Next Entry points to the next empty position in the queue. As nodes are added to the layer, they are placed on the spawning queue at the Next Entry point and the Next Entry point is incremented to an empty position in the queue. The spawning queue thus grows from the bottom. As growth of the layer proceeds, the topmost node in the spawning queue is removed from the queue and used as the jumping off point for further growth. The growth algorithm then enters a loop in which each node in the spawning queue is processed in turn until the spawning queue is empty.

The processing of the spawning node proceeds on a port by port basis. The action taken depends on the kind of element found adjacent to each port. The identity of that element is obtained from the Layer Topology. If the adjacent element is a remote FTP, the spawning node and the port of the spawning node that faces the FTP is placed on the subscriber list. These ports will be enabled after the layer growth is complete. However, if the adjacent element is a node whose status is idle, i.e. not yet part of the active tree, an attempt is made to set up a functional link to that node (which is referred to as the target node). If the attempt is successful, the target node is placed at the end of the spawning queue. Creating such a link requires that a port of the spawning node and a port of the target node be

enabled; the spawning node is enabled first. If the attempt to enable the link between these nodes is not successful, the nodes are disconnected. If the reason for the failure is the detection of a babbler, this subprogram runs a babbler test to ensure that the attempt to disconnect the babbling node was successful. If it is not, an exception is raised which causes the growth to begin again from the start. If the attempt to enable the link is not successful, the link is left in an disconnected state and the status of the corresponding two ports are marked failed. If the link is connected successfully and the Diagnostics Option is not selected, the target node is added to the spawning queue, its status is marked active, and the status of the ports connecting the spawning node and the target node are marked active. Additionally, the updated configuration of the spawning and target nodes is recorded in Layer Configuration (the configuration of the spawning node's port is marked outboard and target node's port is marked inboard, reflecting the flow of data with respect to the ICIS). However, if the Diagnostic Option is selected, the target node is subjected to a set of diagnostic tests which it must pass before being added to the spawning queue. These tests are described in Section 5.4.5. If it does not pass these tests, the status of the target node is marked failed. If it passes the diagnostics, it is added to the spawning queue and the various status fields are updated as before. When all ports of the spawning node have been processed in this way, the next node in the spawning queue becomes the new spawning node. Layer growth continues until the spawning queue is empty.

As mentioned above, this algorithm detects and isolates babbling layer components, thus making it a useful backup tool for layer maintenance. When a port of a spawning node adjacent to a babbler is enabled, the babbler is detected, because its babbling transmissions interfere with the spawning node's status report (which is sent following the node's reconfiguration). Following the detection of the babbler, the spawning node is sent another command instructing it to disable the port adjacent to the babbler, thus isolating the babbler from the rest of the properly functioning layer. The method works because the layer links are full duplex and the reconfiguration command will reach the spawning node through the data line not corrupted by the babbler.

5.4.5

Process Name: Diagnostic Testing

Inputs:
- Node Under Test
- Inboard Port of Node Under Test
- Layer Identifier
- Layer Topology
- Layer Status
- Layer Configuration

Outputs:
- Layer Status
- Layer Configuration
- Passed Diagnostic Tests

Requirements Reference: IC Network FDIR Functional Requirements, Section 5.2.3.3

Notes: None

Description:

For each port of the Node Under Test that is adjacent to an idle node, a series of fault detecting diagnostic tests is performed. The tests are sequential in nature, and if any test fails, the remaining tests in the sequence are not performed. The first test determines if the link between two nodes can be activated. The second test determines whether or not the adjacent node transmits on a port that has been disabled. The third test determines whether or not the Node Under Test retransmits a message on a disabled port. If this suite of tests is completed without any errors, the last test is performed. This final test determines if the Node Under Test responds to the address of another node in the layer.

The first test is performed by using a reconfiguration chain to establish a link between the Node Under Test and an adjacent node. If the attempt to enable the link is successful, the link is left in the enabled state so that the next test can be executed.

In the second test, a configuration command is sent to the adjacent node, utilizing the link enabled during the first test, instructing it to disable all its ports. The node protocol is such that it will carry out this command before transmitting a reply. A properly functioning node transmits a reply from all enabled ports to every command it receives. Since no ports are enabled, this message should not be transmitted. Thus, the node passes this test if a reply to the command is not received. If the node sends a reply, it is considered failed and its status is marked accordingly. Prior to starting the third test, the adjacent node has all its ports disabled.

In the third test, a chain of three transactions is transmitted on the layer. The first transaction is sent to the Node Under Test commanding it to disable all of its ports except its inboard port which connects it to the established layer. The second transaction is sent to the adjacent node commanding it to enable the port facing the Node Under Test for one transmission only. The third transaction is sent to the Node Under Test asking for its status. If the Node Under Test is functioning properly, it will not retransmit any messages, including the command making up the second transaction, to the adjacent node. On the other hand, if it is has failed such that it does retransmit a message on a disabled port, the adjacent node will send a reply which may or may not be transmitted back to the ICIS. In either case, the transmission of this reply will cause the the Node Under Test's valid message detector for the port facing the adjacent node to record the transmission and this information is returned as part of its status message. The Node Under Test passes this third test if the status indicator for the port in question shows no activity and a valid

message was not received. If the Node Under Test fails, its status and the status of its ports are marked failed.

When the above three tests have been performed for every port of the node under test adjacent to an idle node, the node under test is configured so that only its inboard port is enabled. It is then ready for the last test.

In the last diagnostic test, each node in the layer is commanded to report its status, whether or not it is in the active tree. If an unconnected node (i.e. one which is not on either the spawning queue or the active node list) responds to this command, the most recently connected node is answering to this address. This newly added node is then disconnected from the active tree by disabling its inboard port. Furthermore, its status in Node Status is marked failed, since the address decoding function of a node is faulty. It is also possible that a previously connected node could respond with errors. This means that either this node has failed or the most recently added node is talking out of turn. This last added node is then removed from the layer as described above. The node or nodes which had errors on the previous test are again queried for status. If the error indicators are gone, it confirms the talker out of turn hypothesis, and the status of the removed node is set to failed. If not, it indicates that a failure has occurred during the growth process. In the former case, the growth process is continued. In the latter case, the growth process must begin again from the start.

5.4.6

Process Name:	Adding Remote FTPs
Inputs:	Layer Identifier Layer Topology FTP Subscriber List Layer Status Layer Configuration
Outputs:	Layer Status Layer Configuration
Requirements Reference:	IC Network FDIR Functional Requirements, Section 5.2.3.3
Notes:	None

Description:

The ports adjacent to the FTP subscribers are enabled one at a time. Since an FTP which is facing a disabled port will not detect any layer activity, it may be attempting to use the layer

at the time the port is enabled. This could result in errors being detected in the node's reply to its configuration command. Therefore, errors in the node status, which is returned after enabling the root node port of a FTP, are ignored. To verify that the FTP is not babbling, however, the manager must ask for a status read of that node. If the FTP is babbling, that port is returned to a disconnected configuration. This phase of layer growth is complete when all the ports on the subscriber list have been enabled and verified for proper functioning. The Node Configuration and Node Status databases are updated after each reconfiguration transaction is executed and confirmed.

5.4.7

Process Name: Layer Maintenance

Inputs:
Layer Identifier
Layer Topology
Layer Status
Layer Configuration
Error Report

Outputs:
Layer Status
Layer Configuration

Requirements Reference: IC Network FDIR Functional Requirements, Sections 5.2.3

Notes: None

Description:

The various services provided by this process are invoked by the IC Layer Manager. The services provided are: status collection from the nodes in the layer, fault analysis, and layer reconfiguration. If errors are detected during the IC Communications, the IC Network Manager takes that layer out of service and allows the Layer Manager to have sole access to the layer until it has restored full service to all non-failed layer nodes and subscribers.

5.4.7.1

Process Name: Layer Status Collection

Inputs:
Layer Identifier
Layer Topology

Outputs: Status Collection Report

Requirements IC Network FDIR Functional Requirements,

Reference: Section 5.2.3

Notes: None.

Description:

The Layer Status Collection process is the fault detection mechanism of the Layer Manager. When this subprogram is called, it is assumed that the Layer Manager is in control of the interface to the layer, i.e. that the IC Network Manager has taken the layer out of service. In addition to collecting status from each non-failed node in the layer, this subprogram performs some preliminary analysis of error information. This information is obtained by the ICIS when it executes the status collection chain.

The Status Collection Report provides the Layer Manager with a summary of the error information obtained from a preliminary analysis of the response data from the status collection chain. When the ICIS transmits messages on the layer to a node, it observes aspects of the communication and records those observations in registers and buffers for later processing. This constitutes a first stage of fault detection, and it includes detection of: the failure of a node to respond to a command in a reasonable length of time, the presence of transmission errors on the layer, an incorrect number of bytes in a response, and other violations of the communication protocol. In addition to detecting errors on transactions to individual nodes, the overall performance of the layer is monitored for failures which impede the proper functioning of the contention sequence. These failures include a babbler which is flooding the bus with meaningless signals and a data line which is holding the layer in a "stuck on one" condition.

The Status Collection Report presents a summary of the information provided by the ICIS with conclusions drawn about the following error conditions: an interface failure, a babbler, and individual errors detected for each node. If the summary reports that an interface failure has occurred, it also states whether the cause is a failed ICIS or a failed channel connected to the active ICIS. If the summary reports that a babbler is present on the layer, it also specifies whether the babbler was detected during contention for the layer or during data transmission. When either of these errors are present, no further data is provided since the integrity of this data is in question. Furthermore, the Layer Manager's strategies for reconfiguring the layer to eliminate these problems do not require information about individual nodes. Finally, if neither an interface failure or a babbler is detected, an error indicator is provided for each active node in the layer. This error indicator simply notes that an error has occurred. The error could be due to a variety of causes, including a no response error, an HDLC protocol violation, or a check sum error. The type of error is logged in the IC Layer Error Log, however, it is not passed back to the Layer Manager, since its logic does not require this level of granularity to correctly reconfigure the layer.

5.4.7.2

Process Name: Layer Fault Analysis

Inputs:
Layer Identifier
Layer Topology
Layer Configuration
Status Collection Report

Outputs: Error Analysis Report

Requirements Reference: IC Network FDIR Functional Requirements, Section 5.2.3.1

Notes: None.

Description:

The purpose of this process is to analyze the data provided by the Status Collection Report in order to identify both the type of fault responsible for the errors and, if possible, the layer element itself. Two types of analysis are performed: data analysis and error analysis. Each is described in this section.

The data is screened for errors first by the data analysis procedure and then by the error analysis procedure. In some cases, the analysis of the fault is not completed by these procedures, and additional information is necessary before a final conclusion can be drawn. In such cases, the analysis is continued by the Layer Reconfiguration function. This process is discussed in detail in Section 5.4.7.3. That section also describes the actions taken by the Layer Reconfiguration algorithm in response to the various conclusions arrived at in the layer fault analysis process that is discussed here.

Data analysis is the process whereby the status information returned by the nodes is reviewed for the purpose of extracting information about faults in the layer. This process examines the information in the Status Collection Report to determine if a node is transmitting on a port that should be disabled. The transmission may be simple, random noise or a valid message retransmitted by a disabled port due to some fault in the node hardware. This is detected when a node records any activity on the layer (i.e., a change in voltage from low to high or vice versa) or the reception of a valid transmission by a non-failed port which the Layer Configuration shows to be disabled or idle. (Adjacent ports are always in the same configuration, either both enabled or both disabled. They also have the same status, either both active, both idle or both failed.) Initially, this algorithm decides if a babbler is present, because a babbler causes the network nodes to return invalid data. If such a fault is not detected, then this process analyzes the data from the error-free node responses, determines the nodes that have received port activity, and examines the origin of

these messages. If a non-failed, disabled port reports the reception of a message, the node adjacent to that port is transmitting on a disabled port. If a node is located with such a fault signature, then the fault is attributed to the node transmitting on the disabled port and the error report indicates this fact along with the ID of the faulty node. In contrast, if more than one node is found to have this fault, a report indicating an unsuccessful analysis is returned by this procedure. Finally, if such a fault is not present in the layer, the report that is returned by the analysis indicates no data errors were found.

Error Analysis is the second procedure in layer fault analysis. As its name implies, error analysis is the process of deducing which layer element produced the set of errors recorded in the Status Collection Report. Of course not all sets of errors are amenable to analysis. The input space of this subprogram has many combinations which do not pinpoint a specific layer component as being faulty. In these cases, the subprogram returns a value of undiagnosable errors. Furthermore, the assumption underlying all the deductive reasoning in the error analysis is that only one component has failed, and this failure gives rise to all the error symptoms.

If the Status Collection Report indicates that an interface failure has occurred, the error analysis report attributes the errors to a root link failure, indicating the root link which failed and the cause of the failure, either failed ICIS or failed FTP channel. In a similar manner, if a babbler is reported, the error analysis report attributes the errors to a babbler. If neither of these errors is present, the analysis proceeds with an examination of the errors attributed to non-failed nodes in the layer.

If all the nodes in the layer have errors, the error analysis report attributes the errors to a root link failure, indicating the cause as a failed ICIS. If some nodes have errors and some do not, two possible failure modes are considered: (1) a failed link (or node) through which no transmission takes place or (2) a single node failure. The single node failure symptom could be indicative of a node which does not respond to commands but which continues to retransmit messages as it did before the failure. It could also be a node which itself is not failed but to whose address another node in the layer responds. The single node failure is easy to diagnose since exactly one node in the Status Collection Report shows an error. The reconfiguration strategy used in this case is described in Section 5.4.7.3.

If two or more but fewer than all nodes have errors, the remaining problem is to determine if the cause of those errors is a link or node whose transmission/retransmission function is no longer operational. The basic idea is that when a link or a node fails in this way, then all nodes downline of this fault also have errors. The signature of such a failure is these nodes form a treelike pattern in the layer. It should be noted that another failure mode which would produce a similar pattern of errors is a node which babbles on all its outboard ports. To determine if the observed errors fit this case is a three step process. The first step is to identify a node which qualifies as the root of the failed tree. Such a node is a node which had errors itself but which has an inboard port (the port which receives

commands sent by the ICIS) adjacent to a non-failed node. To verify this fault hypothesis, exactly one such node should have this characteristic. If more than one such node exists, the fault is considered undiagnosable. However, if a root is found, the second step is to determine whether or not all nodes downline of the root have errors attributed to them. This is accomplished by a recursive subprogram. The subprogram accepts a node as a parameter; the node is referred to as the current node. The first call to the subprogram uses the root of the failed tree as the input parameter. The subprogram examines the nodes adjacent to the outboard ports of the current node. If any of these nodes does not have errors attributed to it, the subprogram returns a value of false and the fault is considered undiagnosable. However, if a treelike pattern is established, the third step of the pattern checking process can proceed. This step verifies that all nodes which had errors appeared in the failed tree, i.e. no nodes with errors lie outside the tree. If nodes with errors are found outside the tree, the fault is considered undiagnosable. If all three steps in the process support the failed link/failed node hypothesis, an error analysis report is returned stating the fault is a failed link or a failed node. Additional information contained in the report is: the node number of the failed root of the tree, the port number of the inboard port of this node, and a list of nodes in the tree. The final determination of whether or not the fault is due to a failed link or a failed node is made during layer reconfiguration.

5.4.7.3

Process Name:	Layer Reconfiguration
Inputs:	Layer Identifier Layer Topology Layer Configuration Layer Status Error Analysis Report
Outputs:	Layer Status Layer Configuration
Requirements Reference:	IC Network FDIR Functional Requirements, Section 5.2.3.2
Notes:	None.

Description:

The purpose of this process is to reconfigure the layer so as to restore error free communication to all reachable, non-failed nodes in the layer. The action taken by this process will depend upon the type of failure reported in the Error Analysis Report. The fault identified in this report is actually a speculation about what is causing the errors on the layer. This process in effect tests this hypothesis and then verifies that the layer is again

fully operational. Thus the layer may go through several intermediate configurations before the reconfiguration process is complete.

There are five classes of faults identified by the Layer Fault Analysis process described in Section 5.4.7.2. They are a babbler, a link or node failure, a node which transmits on a disabled port, a single node failure, and an undiagnosable failure. The Error Analysis Report indicates which one of these failure modes is presently causing disruptions on the layer. Depending on the type of fault, it may also contain some additional information about the the source of the problem. A separate strategy exists to deal with each of these fault classes.

The reconfiguration process is considered complete when the node status chain is executed on the reconfigured layer and does not detect any errors. Further, the backup stratagem for dealing with error phenomena which occur during a reconfiguration attempt but which are not anticipated is layer regrowth.

The reconfiguration strategies are designed to deal with both active and passive faults. Passive faults are characterized by the non-retransmission of data, sort of a barrier or obstacle to data flow in the layer. A disconnected cable is an example of such a fault; data cannot be retransmitted over this cable but transmission between other connections in the layer is not affected. Active faults are characterized by the disruption of data flow in the layer beyond the boundaries of the failed component itself. An ICIS with a transmitter stuck on high is an example of this type of fault; the stuck on condition is retransmitted throughout the layer, possibly disrupting transmissions between all layer connections. Since the same error conditions generated by a broken root link could also be generated by an ICIS stuck on high, the reconfiguration algorithm must identify the specific cause of the problem so as to effect a repair.

When a babbler is detected in the layer, the layer is regrown without the diagnostic option since the detection and isolation of a babbler does not require any diagnostic testing.

A subprogram called Repair Link or Node Failure is called to handle layer reconfiguration when the Error Analysis Report indicates the presence of a failed link or node. Since a failed node generates the same error pattern as a failed link, this subprogram must determine which fault has actually occurred and reconfigure the layer accordingly. The Error Analysis Report contains the node number of the node suspected to be failed, its inboard port, and a list of nodes which are unreachable as a result of this failure. It is first assumed that a link has failed. The failed link is disconnected, and an attempt to reach the failed node, i.e. the node immediately downline from the link, is made by using any spare ports on that node which are adjacent to non-failed nodes. The chain used to reconnect this node to the rest of the layer contains three transactions. The first two transactions enable the ports on either side of the new inboard link; the third transaction disables the former inboard port of this node in case the node adjacent to that inboard port is a babbler. If this strategy fails to restore communication with the failed node (possibly because no spare

ports are available), data is assembled which will allow each branch of the failed tree to be reconnected to the active layer. This data consists of a list of nodes for each branch stemming from the failed node (i.e. a separate list for each set of nodes which lie downline of each of its outboard ports). Only one successful connection to any spare port on a branch is necessary to restore communication to the entire branch (and possibly to the failed node and all other nodes in the failed tree). Again a three transaction chain is used, this time for a different purpose. The first two transactions enable the ports on either side of the new link while the third transaction attempts to obtain status from the failed node. If the failed node correctly returns its status, the repair is complete and the absence of errors is verified by collecting status from every node in the layer. If the failed node is still not reachable, the port connecting this node to the present branch is disconnected and the proper functioning of the newly enabled link is verified. Then all nodes on this branch are removed from the failed node set. The net effect of this process is to restore communication with all reachable nodes in the layer while isolating the failed node. As communication to each branch is restored, the possible pool of spare links increases. Thus if any branch was not connected because of a lack of spare links, this branch is retried whenever a connection to another branch is successful. Any nodes which are still unreachable at the end of this exhaustive process are assigned a status of failed.

If a node retransmits valid data on a port which should be disabled, the node must be removed from the layer. This failure mode is distinguished from a babbler which is always transmitting a random bit stream or is stuck on one. When a babbling port is identified, the adjacent port of the neighboring node is disabled. This neighboring node will not retransmit on its other enabled ports anything received by the disabled port. Furthermore, the node will ignore any random bit patterns it receives. However, if the neighboring node receives a request for its status on a disabled port (as might occur if a failed node is transmitting on a disabled port), it will transmit its status on all of its enabled ports. If this failed node is not removed, each time the Layer Manager asks for status from the node adjacent to this port, it would receive two valid commands to report its status, which will result in two status responses. However, only one response is expected. Once the first response is received by the Layer Manager, another node will be commanded to report its status. The second response of the node may interfere with the reply of a node whose transaction is later in the chain, making it appear that this next node has failed to respond correctly to a command. Once the failed node has been removed from the layer, status is collected from the remaining nodes to verify that in fact the fault has been identified and isolated. If errors are still detected in the layer, a full regrowth, with a complete set of diagnostic tests, is performed.

The subprogram which removes a node from the layer is called Remove Failed Node and Reconnect to Trees. As the name implies, removal of a node is a simple matter if the node is a leaf; only the link connecting it to the layer needs to be disconnected. This is accomplished with one chain. However, if the node is the root of a subtree in the layer, the

nodes downline from the failed node need to be reconnected to the layer through alternate links.

Prior to beginning the reconfiguration of the layer, the nodes downline of each outboard port of the failed node (i.e. the nodes on each branch of the tree emanating from the failed node) are added to a reconnection queue. Each of these nodes is also added to a set of unreachable nodes. The link connecting the inboard port of the failed node to the rest of the layer is then disabled. Next, a loop is entered in which an attempt is made to reestablish a connection to each isolated branch via a spare link from a node which is still reachable, i.e. is not a member of the unreachable node set. Only one such connection needs to be made to restore communication to all the nodes in the branch. After the new connection is enabled, the link connecting the failed node to this branch is disconnected. As each branch is reconnected, the nodes in that branch are removed from the failed node set. If any branch is successfully reconnected, the branches which were not connected during earlier attempts are tried again since more spare links become available. Thus this algorithm, while isolating the failed node, restores communication to every reachable node in the layer. Nodes which cannot be reached because earlier failures are marked failed.

If a single node in the layer has errors, the reconfiguration is handled by a subprogram called Reconnect, Remove, or Regrow. This failure can occur: (1) if the failed node is a leaf node, (2) if its retransmission function still works correctly but its status reporting capability is impaired, or (3) if another node is responding to this node's address making it appear that this node is failed. The failed node is isolated from the layer, as described above in the discussion of Remove Node and Reconnect to Trees. However, care is taken not to address this node directly. When the node is isolated, this node is again queried for its status. If a valid response is received, indicating the presence of a node which responds to the addresses of other nodes, the layer is regrown with a full set of diagnostic tests to isolate this faulty node. Otherwise, an attempt is made to find an alternate root to this node using any port except its previously failed inboard port. The configuration command sent to this node as part of the link enabling procedure will disable this failed inboard port.

If the attempts to reconfigure the layer have not succeeded in eliminating errors, then a layer regrowth is performed. This is the back up reconfiguration strategy, used when all else fails.

Following the reconfiguration of a layer, the Layer Status is updated to reflect the current state of the layer hardware. If any nodes have been isolated from the layer as a result of the reconfiguration, the transaction for that node is removed from the status collection chain. Additionally, the Layer State is given the value Repaired.

6.0 THE PHYSICAL AND DATA LINK LAYERS

The Physical Layer of the ISO Model is concerned with the electrical, functional, and procedural characteristics necessary to establish, maintain, and disconnect a physical circuit. The Data Link Layer is concerned with the low-level requirements (i.e., communication protocol, fault detection, and fault recovery) necessary to send valid blocks of data over a physical link [4]. In this section, the Physical and Data Link Layers of the AIPS Distributed Engineering Model are described.

6.1 Functional Requirements

A communications interface is required to permit a physical link between two or more FTPs in the distributed AIPS system. This interface provides a communication protocol to allow data to be reliably transmitted between a set of subscribing network sites. Furthermore, this physical and data link interface is designed to be a modular redundant architecture in order to enable fault tolerance and to facilitate reconfiguration. The hardware implementation also provides the necessary status information permitting fault detection and isolation operations on the communication process.

6.2 Hardware Specifications

A hardware interface was developed specifically for the AIPS project to provide the required Physical and Data Link layer functions. This interface between the FTP and the Inter-Computer network is referred to as the Inter-Computer Interface Sequencer (ICIS). The ICIS is responsible for managing the low-level bus protocol and data formatting details in an autonomous manner, thus relieving the processors of the FTP of the task of managing the IC network at microsecond time frames. Each channel of an FTP has one ICIS which is accessible to both processors of the channel. The physical connection between the ICISes of two separate FTPs is the Inter-Computer Network as illustrated in Figure 5-1. As described in Section 5.0, the Inter-Computer Network is comprised of three layers. As shown in Figure 5-1, each channel's ICIS has an individual interface to each of the three layers. When the IC Communication Services on a triplex FTP transmits a message, three copies are transmitted (at essentially the same time); one copy on each IC layer. The ICIS in any one channel of an FTP listens to all three network layers but can only transmit on one. The transmission function of the ICIS is responsible for serializing data, resolving network contention, and transmitting data on the network. In the other direction, this network interface listens to three bit streams, deskews them, and stores all redundant data streams, along with certain status information, in a dual-ported memory accessible by the channel's processors.

The ICIS hardware specifications are presented in the following sub-sections. First, a description of the Physical and Data Link implementations is given. These implementations are completely independent of the redundancy aspects of this communications interface.

Second, the method of arbitrating for network possession is described. Finally, the interface between the ICIS and the core FTP is discussed.

6.2.1 Physical and Data Link Layers

The IC network Physical Layer uses 2 Mbit/Second NRZI (Non Return to Zero Inverted) bit serial protocol. In this protocol, a logical '0' is represented by a transition, either from a high to a low or from a low to a high, and a logical '1' is represented by a lack of a transition. A constant high level or a constant low level represents a string of '1's on the network.

The Data Link Layer protocol used in the Distributed AIPS Engineering Model hardware is a 2 MHz HDLC protocol. In this protocol, no more than five '1's can be transmitted in a row. A '0' is inserted after five '1's by the HDLC transmitter chip, and it is subsequently deleted at the receiving end by its counterpart. Six '1's in a row constitute an HDLC flag which is used to mark the beginning and the end of an HDLC frame. Seven '1's constitute an HDLC abort signal and more than 7 '1's represent an HDLC idle state. The last two HDLC signals are not used in the AIPS Model. At 2 MHz, each bit width is 0.5 μsec, which makes the maximum time the network would stay high or low while transmitting HDLC data 3 μsec.

The HDLC protocol provides automatic address detection, embedded control information, and cyclic redundancy checking to detect transmission errors. An HDLC frame contains an opening flag, address byte, control byte, data bytes (in AIPS up to 119), two FCS (Frame Check) bytes, and a closing flag. The opening and closing flags are identical and consist of a '0', followed by six '1's and a '0'. It is not possible for a flag to look like data since the HDLC protocol specifies that within the data field after five continuous '1's a '0' is added. Additionally, the HDLC address detection mechanism can be configured to allow the receiving sites to accept only those data packets which are addressed to the FTP or which are "broadcast" messages (data which is sent to all subscribers on the network).

6.2.2 Inter-Computer Network Contention

This section describes the design and implementation of the bus contention protocol that is used to arbitrate access to the inter-computer network. The issue of reliably resolving bus contention is critical in a mixed redundancy distributed system such as AIPS.

The inter-computer circuit switched nodal network is triplicated in the AIPS Distributed Engineering Model. The three layers of the network are used to provide redundancy rather than maximize bandwidth. In normal operation, the three layers carry identical information. By making redundancy management and network contention transparent to application software, a very simple and friendly virtual architecture, that of a highly reliable

simplex bus, is presented to the applications programmer. In fact, most aspects of the network contention are hidden from the system software as well.

For access arbitration purposes, the triplex network is treated as a single entity. FTPs, regardless of their redundancy level, compete for all three layers of the network. At the end of the contention sequence one, and only one, FTP may have access to all three layers of the network. Thus, if a duplex FTP wins contention, it is given exclusive use of all three network layers even though it can transmit on only two of the three layers. No effort is made to maximize the network bandwidth by providing simultaneous access to a duplex FTP on two layers and a simplex FTP on the third layer, for example.

The bus arbitration scheme must meet certain requirements for the type of applications for which the AIPS architecture is intended. Communication between critical functions which are resident in triplex FTPs must not be interrupted or corrupted by lower criticality functions resident in duplex and simplex FTPs. Triplex FTPs should be given access priority over all others. Similarly, duplex FTPs should have priority over simplex FTPs.

In a distributed system such as AIPS, the contention resolution must be fair and equitable to all sites of like redundancy. Over a period of time, for example, all triplex FTPs should have an equal chance of getting network access. Similarly, all duplexes should be served equally well by the network as should all simplexes. However, the arbitration scheme should also be flexible enough so that a low criticality function that may be assigned to run on a triplex FTP should not hog the network. A triplex FTP should contend as a duplex or simplex, if the function requesting the communication is of appropriate low priority.

No single point failure should result in a communication disruption between two triplex computers. The arbitration logic must be able to resolve bus contention in a reliably robust manner even in the presence of an arbitrary fault. In other words, a malicious failure in a simplex FTP or in one channel of a redundant FTP should not be able to disrupt traffic on more than one layer. Furthermore, in keeping with the spirit of the distributed nature of the AIPS architecture, the arbitration authority must not be centralized. It should be distributed throughout the system, and all processing sites wishing to access the network at any given time should arrive at a consensus about the sole winner cooperatively but independently and in a fashion which preserves network integrity in the face of failures or damage.

In addition, redundant channels within an FTP must come to a consensus as to whether or not they are contending for the bus and at the end of the contention sequence, whether or not they have won access to the bus. The bus contention protocol does not occur in the FTP processors themselves but rather in the dedicated Inter-Computer Interface Sequencer.

6.2.2.1 The Laning Poll

The IC bus arbitration protocol is an evolution of the Laning Poll, a protocol used previously to resolve bus contention internally in the Fault Tolerant Multi-Processor (FTMP).

The Laning Poll is a bit serial algorithm for prioritized contention of serial buses. The Laning Poll assumes that multiple sites can transmit on the same bus or serial line simultaneously, each site then receiving the 'OR' of all bus transmissions. That is, if any site transmits a '1' all sites will hear a '1'. Each site contending for the bus has its own unique binary priority vector P(v1,v2, ..., vn). A higher number signifies a higher priority. The Laning Poll algorithm guarantees that the site with the highest priority will win the bus. The poll consists of sequentially transmitting each priority bit (from most significant to least significant) on the bus. Each site behaves according to the following algorithm during the polling period.

For all i while still contending do:

Transmit P_i on the serial bus
- If $P_i = 1$ and Received value = 1 then continue
- If $P_i = 1$ and Received value = 0 then win bus
- If $P_i = 0$ and Received value = 1 then quit
- If $P_i = 0$ and Received value = 0 then continue

This algorithm requires that all sites on the bus be synchronized in some manner prior to the start of the poll so that all sites are transmitting their ith priority bit simultaneously. There is no global clocking mechanism to synchronize all sites on the bus and, in fact, there is no need to synchronize the processing sites except for the bus arbitration poll. Therefore, the processing sites operate asynchronously until a site needs access to the bus. A start bit precedes the polling sequence to ensure synchronization of the polling sequence across all sites, as further explained in Section 6.2.2.2.1.

6.2.2.2 The AIPS Contention Protocol

The AIPS contention protocol uses a modified form of the Laning Poll. It consists of two parts, the redundancy contention sequence and the priority contention sequence. The redundancy code sequence consists of 3 bits: S, T, and D (denoting Start, Triplex and Duplex, respectively), and the priority code sequence consists of three FTP priority bits followed by six FTP ID bits. The objective of the redundancy contention sequence is to resolve contention between the different levels of redundant elements contending for the bus (i.e., triplex, duplex, simplex). At the end of this sequence, all non-failed FTPs still contending should be of the same redundancy level. The priority contention sequence resolves contention among non-failed FTPs of the same redundancy level according to the priority and the ID bits.

6.2.2.2.1 The S Bit of the Redundancy Code Sequence

The bus contention begins with the 'S' (Start) bit. A FTP may initiate a contention sequence if the network has been idle for greater than 512 μsec. To initiate a contention poll, a FTP transmits a '1' on all layers to which it is connected. Other FTPs may join in the poll sequence by transmitting their own S bits while the initiating FTP is still sending its S bit. All channels OR the S bits on all three network layers. The S bit serves to synchronize all contending sites at the beginning of the poll sequence.

6.2.2.2.2 The T Bit of the Redundancy Code Sequence

All triplex FTPs contending for the network transmit '1's on all three network layers during the 'T' (Triplex) bit. All FTPs contending for the network vote the T bit on the three layers. If the voted result is a '1', indicating the presence of a triplex in the contention sequence, all FTPs not configured as a triplex drop out of the contention sequence. All contending FTPs configured as triplexes skip the next redundancy code sequence bit and proceed directly to the priority contention sequence. If there is only one triplex in the contention, it should obtain a voted result of '0' and declare itself the winner, and it does not go through the rest of the contention sequence. A triplex not contending as a triplex for low functional priority reasons does not transmit during this bit. If duplexes or simplexes obtain a voted result of '0' during the T bit, they conclude that no triplexes are contending and they go on to the D bit.

6.2.2.2.3 The D Bit of the Redundancy Code Sequence

The 'D' (Duplex) bit is used to resolve contention between duplex and simplex FTPs. During the D bit poll, a FTP configured as a duplex transmits a '1' on the two layers to which it is connected. All contending duplex and simplex channels OR the three network layers. If the result is a '1', indicating the presence of one or more duplexes, all FTPs not configured as duplexes drop out of contention. All FTPs configured as duplexes proceed to the priority contention sequence. If there is only one duplex in the contention, it obtains a voted result of '0' and declares itself the winner. Further, it does not go through the rest of the contention sequence. If simplexes obtain a result of '0', they proceed to the priority contention sequence.

6.2.2.2.4 The Priority Sequence Bits

The priority sequence consists of 3 priority bits and 6 FTP ID bits. The voted or ORed result of each of these poll bits, as described below, is treated in the traditional Laning Poll manner.

Triplex Contention - Each channel of triplex FTP votes the three network layers.
Duplex Contention - Each channel of duplex FTP ORs the three network layers.
Simplex Contention - Each simplex FTP channel ORs the three network layers.

6.2.2.3 Implementation of Network Contention Hardware

The ICIS is responsible for contending for the network, formatting data, transmitting data from the channels onto the network, receiving data from the network, and storing data in its memory in an appropriate format for use by the channel's processors. Multiplexing between these tasks is the responsibility of the ICIS master microsequencer which obtains its control inputs from the processors (through a control register) and a sequence of microinstructions. Only those parts of the ICIS design that pertain directly to the implementation of the contention scheme will be discussed here.

Central to the operation of the ICIS contention circuit is a network activity monitor. Each ICIS in each FTP channel contains three identical monitor circuits, each dedicated to a single network layer. The individual states of the three network layers, as determined by the network activity monitors, are combined within each ICIS to determine the overall state of the IC network. A microsequencer in the ICIS uses the network state in combination with control inputs from the processors to determine its next action.

The three ICISes in the three channels of a triplex FTP operate in tight synchronism with each other in order to assure identical operations in the three channels and on the three network layers. However, due to the lack of a system-wide synchronization clock, the initiation of a poll sequence by a FTP is totally asynchronous to all other FTPs in the system. Therefore, it is necessary for the ICISes in an FTP to follow the source congruency principles and make the network state congruent across the ICISes before using that information to determine their next action. This is accomplished by exchanging the internal ICIS states across the three channels using dedicated exchange hardware. The design of this exchange hardware follows the principles that are outlined in the following sections.

6.2.2.3.1 Poll Bit Timing

As stated earlier, the IC network Physical Layer uses the NRZI protocol. Further, the Data Link Layer protocol used in the AIPS Distributed Engineering Model hardware is a 2 MHz HDLC protocol. At 2 MHz, each bit width is 0.5 μsec, which makes the maximum time the network would stay high or low while transmitting HDLC data 3 μsec.

The Laning Poll contention scheme is superimposed on the HDLC protocol by making the poll signals wider than 4 μsec so that they are not interpreted as HDLC signals. Figure 6-1 shows the timing for the poll bits. Suppose that the site labeled 'A' initiates the contention sequence by putting the start bit S on a network layer and is joined in the poll by a second site labeled 'B' which is located the maximum number of nodes away from A. Figure 6-1 shows that the start bit width should be 48 μsec and that all subsequent poll bits should be 24 μsec wide.

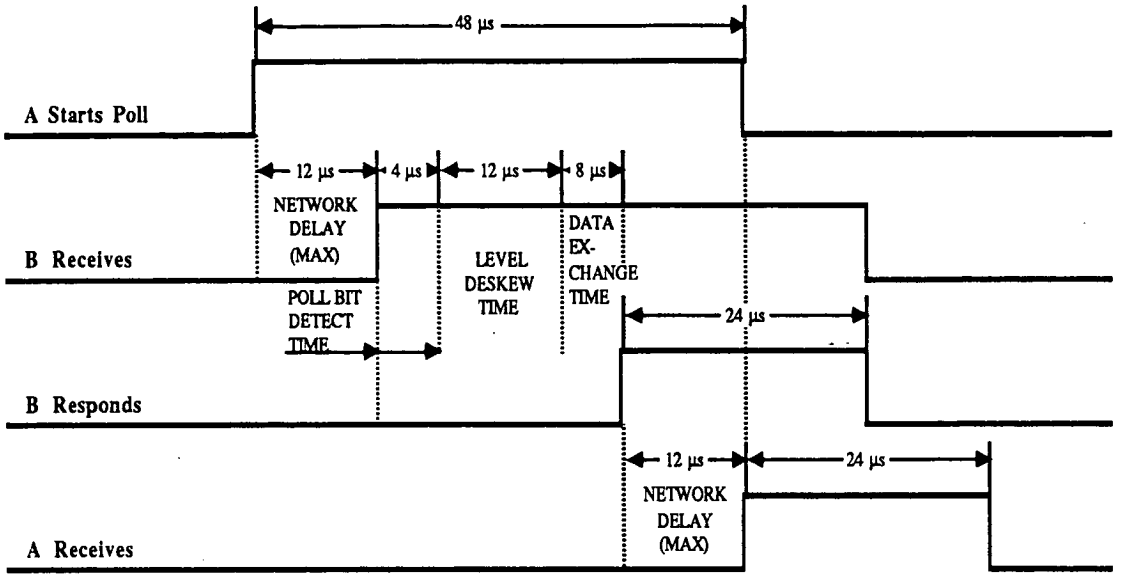


Figure 6-1. Poll Bit Timing

The additional 24 µsec at the beginning of the start bit for the site initiating the contention is used by the receiving sites to identify the poll bit, compensate for skew between network layers, and allow the ICISes time to exchange the poll detection event. A poll detect time of 4 µsec is used to distinguish it from the longest HDLC string of '1's. The network layer deskew time of 12 µsec is required since, in the worst case, the poll bit may have passed through a maximum number of nodes in one layer and a minimum number of nodes in another layer. Because passing through each node produces a delay of about a third of a microsecond in the Engineering Model nodes, a deskew time of 12 µsec would allow more than 32 nodes in each IC network layer. At this point each ICIS determines, by an examination of the network layer activity monitor outputs, if a contention has begun. This information is then exchanged across redundant ICISes and made congruent. This takes two cycles of the fault-tolerant clock or 8 µsec.

6.2.2.3.2 Network Activity Monitors

The function of a network activity monitor circuit is to determine the state of the network layer it is monitoring. Each channel's ICIS has three network monitor circuits. A network layer can be in one of four different states depending on how long it has been High or Low, as defined in the following:

Data: it is carrying HDLC data including flags (it is High or Low for less than 4 µsec).
Idle: it has been quiet (Low) for 512 µsec or more.
Stuck: it has been High for 512 µsec or more.
Poll: it has been High for 4 µsec or more but less than 512 µsec.

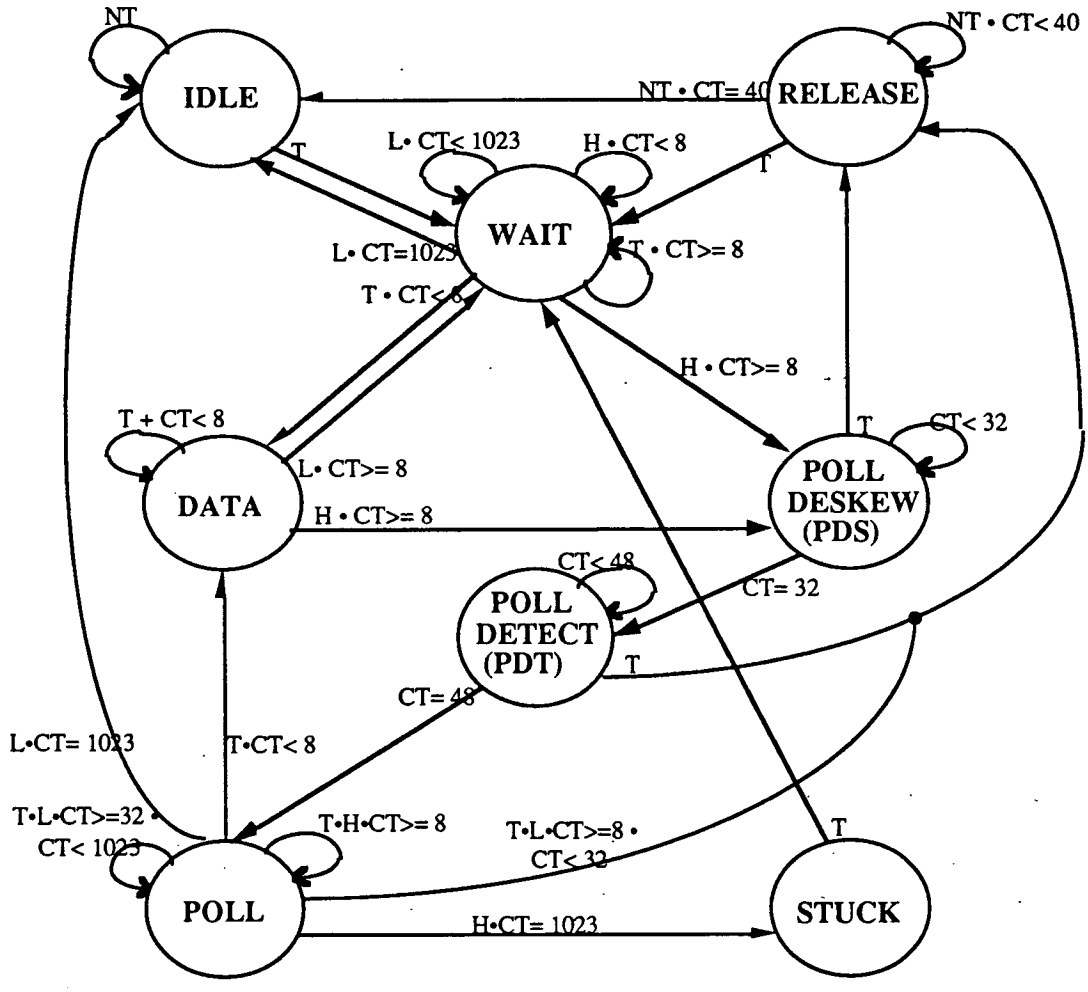
Figure 6-2 shows the state transition diagram for the network activity monitor circuit. The monitor circuit has a total of eight states, four of which correspond directly to the four network states. In addition, there are four other states which are necessary for the monitor to handle indeterminate situations and time skews in the network layers. These latter four states are as follows:

Wait:	network state is unknown.
Poll Deskew:	network transitioned into high state between 4 and 16 μsecs ago.
Poll Detect:	network transitioned into high state 16 μsecs ago. Specifies the point at which the ICIS could join a poll if so instructed.
Released:	a high-to-low transition detected while the monitor was in the first part of a polling sequence.

The network activity monitor circuit is driven by the HDLC clock ticks which occur every 0.5 μsec. It times the intervals between transitions on the network and provides an indication of the most recent activity on the network layer. In Figure 6-2, 'CT' represents the value of the clock tick counter. The 'H' and 'L' that appear on state transitions refer to the network layer states High and Low, respectively, and 'T' is used to indicate a transition of the network layer from a High to a Low or vice versa. The asterisks (*) in the diagram represent the logical 'AND' of two events. The counter CT is reset to zero whenever an T occurs, and rolls over to zero after a count of 1023.

The states Wait and Poll Deskew are necessary to allow for timing skews on the network layers and are used to determine when a transition to Idle or Poll is about to occur. Additionally, the Release state is used to identify the network release function. This happens when the FTP that is controlling the network finishes transmitting data and wishes to release the network for use by other FTPs. The controlling FTP relinquishes the network by initiating a poll and then dropping out of the poll after 16 μsecs. This process initiates a polling sequence among any sites waiting to contend for the network. If one site is waiting to contend, it will win the poll at the first bit. If two or more sites are waiting, the poll proceeds normally. If no sites are waiting to contend, the monitor circuit indicates an idle. The network release function has been included to allow a FTP to give up the network without the requirement that 512 μsec elapse before another FTP can gain control.

Three different circuits in each ICIS combine the states of the individual network layers (as determined by the three monitor circuits) to produce a consistent view of the network status. The three network states of interest to the contention logic are: Data, Poll, and Idle. In addition, certain abnormal conditions can result in an Abort signal. These signals are provided as inputs to the ICIS microsequencer or the ICIS state machine. Figure 6-3 provides a overview of the logic blocks involved in the polling mechanism.



T = Layer input "Toggle" or "Transition"
NT = No Layer input "Toggle"
L = Layer input Low
H = Layer input High
CT = counter value with counter incremented every 0.5 microsecond (wraps around to zero after 1024 counts)
* = Logical AND
+ = Logical OR

Figure 6-2. Network Monitor State Diagram

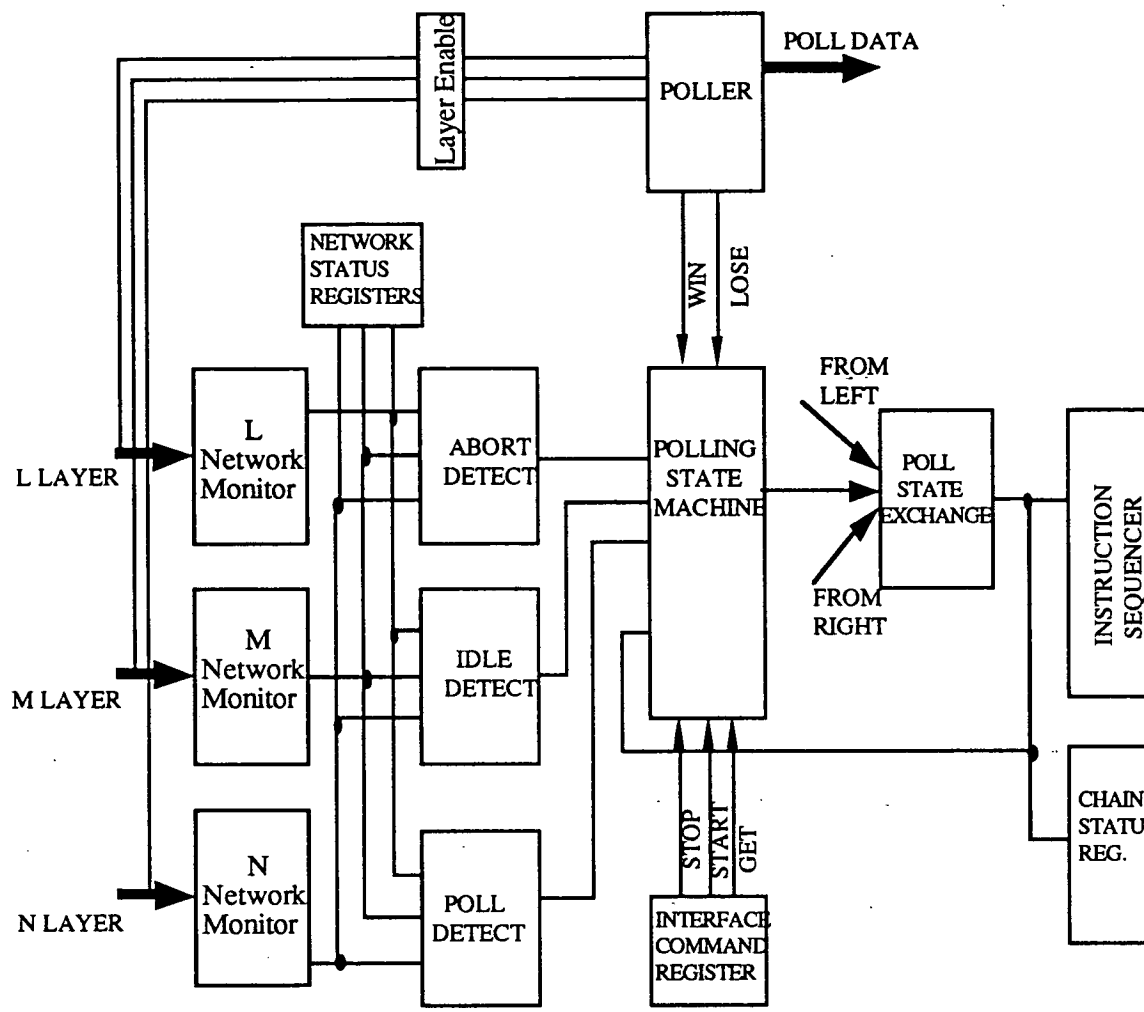


Figure 6-3. Polling Logic Blocks

The Abort/Data Detect circuit produces the Abort signal which is used to terminate a contention. For simplex or duplex sites, data on any layer constitutes an abort. For triplex sites, data received on two layers separated by less than 12 μsec is required for an abort. The Data Detect signal is presented to the circuit which identifies an idle network. This signal, like the triplex FTP abort, requires two layers to indicate data with less than 12 μsec of separation.

The Poll Detect circuit produces a pulse if any network monitor enters the Poll Detect state for the first time and any of the other monitors is in either the Poll Detect, Poll Deskew, Idle, or Released states. This means that 12 μsec after a poll is detected at least one other layer must be either polling or be idle. A majority of layers polling or a single layer polling with the other two layers idle are poll conditions.

The Idle Detect circuit is a set-reset latch which presents the Idle Detect signal to the ICIS state machine. The latch is set to indicate the network is available whenever a layer transitions to the Idle state and at least one other layer is at either Idle or Released. The Idle Detect indication remains until either a Poll Detect or Data Detect signal occurs, as described above.

The three circuits Abort/Data, Poll Detect, and Idle Detect are synchronous with the fault-tolerant clock, and therefore they are synchronized with the ICIS state machine and data exchange as well.

6.2.2.3.3 ICIS State Machine

Figure 6-4 shows the state transition diagram for the ICIS state machine. Its function is to conduct the poll and produce the signals 'Win' or 'Lose' depending on the outcome of the poll. The ICIS state machine gets network state information from the three network activity monitors in the form of the four signals described previously. Additionally, it gets control inputs from the CP or the IOP. The control inputs, in the form of bits in a control register, that are relevant to the contention function of the ICIS are as follows:

- **Get** - When set, the ICIS is instructed to contend for the network until network possession is achieved and to retain possession until the bit is cleared.
- **Cstop** - When set, the ICIS is instructed to return to the not-contending state unless a poll sequence is in progress or the ICIS is in possession of the network. This bit has precedence over Get.
- **Stop** - When set, the ICIS is instructed to return to the not-contending state and to stop the ICIS microsequencer. This bit has precedence over Cstop, Get, and Start.
- **Start** - When a 1 is written to this bit of the control register, the ICIS is instructed to start a contention sequence unless a contention is already in progress. This bit has precedence over Get and Cstop.

The normal control input to get access to the network is Get. Cstop, or conditional stop, is used to implement 'polite preemption'. This allows the processor to request the ICIS to preempt a poll unless one has already begun. The Stop control forces the ICIS to cease all of its activities. The Start control is used by the processor to force a poll regardless of the network state. This is useful when a babbling failure of a FTP may deny network access to other FTPs. Finally, the signal 'Initialize' in Figure 6-4 represents a manual or power-up reset of the ICIS state machine.

The horizontal levels on which the states are shown in Figure 6-4 represent general groupings. The first level is the states for which the ICIS has not been instructed to contend for the network. The next level represents those states for which the ICIS is waiting for conditions to allow a poll. The third level contains states for which the poll is in progress, and the fourth contains those states for which the poll was won. Only in the state 'network possession' is the ICIS assumed to have control of the IC network.

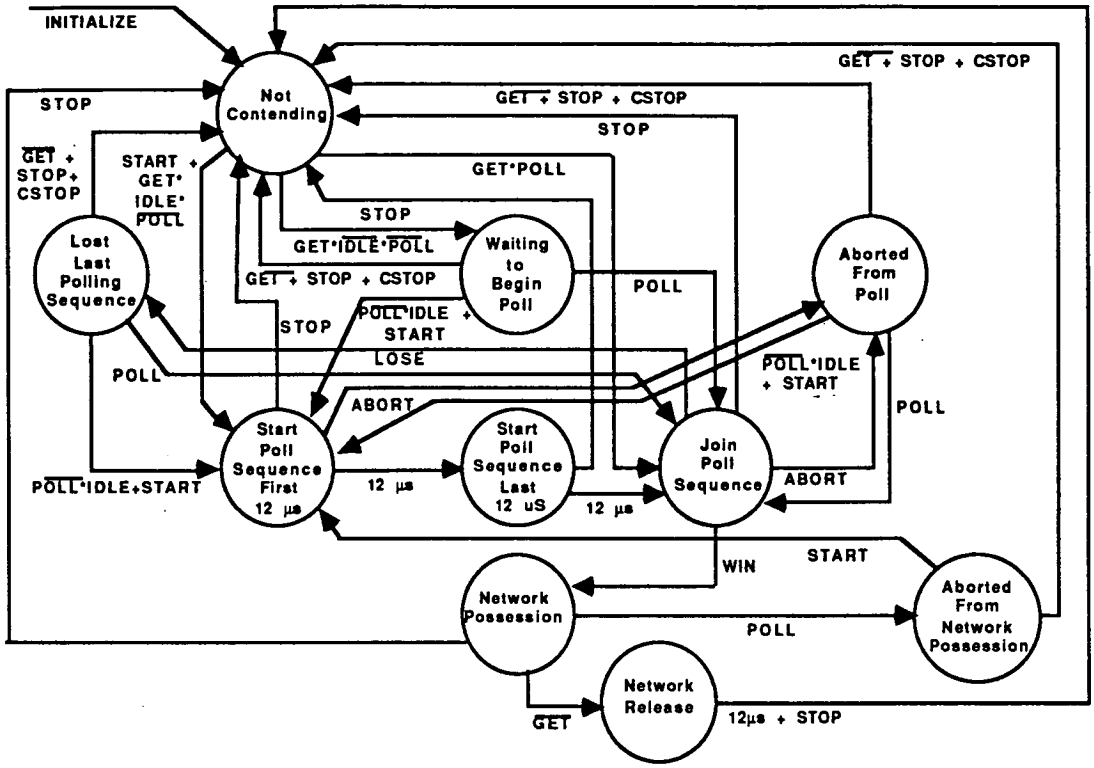


Figure 6-4. ICIS State Machine

The state machine which performs the poll is active during the states labeled 'start poll sequence' and 'join poll sequence'. This state machine continues to poll until the poll is either won or lost. If the polling machine exhausts all bits of the priority and ID without reaching a decision, the poll continues with zeroes sent to the driver so that the ICIS will appear to all other sites as to have dropped out of contention. The poll may be stopped before a decision if an Abort is detected on the network or if the ICIS state machine transitions to another state as a result of a control register signal.

The state vector of the ICIS state machine shown in Figure 6-4 is available for the IOP or the CP to read as a memory mapped register. The processor may then monitor the process of contention and decide to intervene if necessary. The 'bus busy' state can be detected by the processor as a failure of the ICIS to transition from the state 'waiting to begin poll'. If a poll has been aborted, the ICIS records this condition in a register separate from the state vector so that the indication is not be lost at the start of the next poll. The processor is responsible for clearing this bit.

The network release pulse described above is generated by the ICIS as a result of a transition from the state 'network possession' due to the signal GET being set to zero. The other transitions from this state are not the result of normal operation and do not produce the pulse. Should a very long message be transmitted such that a single network possession is insufficient, the ICIS will be instructed by the processor to end data

transmission and start a poll. Under a condition where data is followed by a poll, the ICIS must insert 12 μsec of zeros on the network before driving it with the first poll bit. This insertion of zeros is included so that, at all receiving sites, the skew between network layers cannot cause the first poll bit to be received on one layer while data is being received on the others. Such a condition, a poll bit detected on a single layer and data detected on either of the other layers, is not considered as the start of a poll. This is so that two sites simultaneously transmitting data on the same network layer (e.g., a triplex and a failed simplex) do not cause a third site to detect a poll bit (via its received OR of the two data streams) and thus begin to poll.

6.2.2.3.4 Cross-Strapping of ICIS Channels

One issue important in the design of the contention hardware is the necessity of cross-strapping redundant channels of the contention hardware (ICIS) to each other. This is necessitated by the absence of a means to synchronize the activities of the FTPs. An FTP may begin a poll at any time, provided that the network is in a certain state as described previously. The poll start event is an asynchronous input to all other FTPs. Therefore, it must be treated with the same care that is accorded to any input to the FTP. Specifically, there is asynchronism between the command to contend for the network (from a processor to an ICIS) and the detection of the proper conditions for a poll to begin. Without exchange of information between ICIS channels, it is possible that one ICIS of an FTP would not poll until a subsequent polling sequence, even when no failures are present. This is illustrated in the scenario in which the control bit indicating that an ICIS channel is to contend is set by each IOP of a triplex site just as a poll sequence is detected on the network. Each IOP will set this bit with its local phase of the fault tolerant clock. There may be sufficient skew between clock phases such that two ICISes will have the bit set just before the poll is detected with the third having its bit set just after. The ICISes which are set prior to the beginning of the poll will be able to join in contention, while the ICIS which was set afterward will have decided that it is too late to join that poll and will wait to join the next. Information exchange between ICIS will allow all channels to act in concert.

Figure 6-5 shows the state machines of a triplex ICIS which are associated with the control of network contention. The next state as determined by each ICIS is passed to the other channels of the FTP by a data exchange and then voted to determine a congruent next state. The data exchange guarantees that each ICIS will be voting the same set of next states. Figure 6-6 shows the state exchange for one channel. Sixteen connections in each ICIS for each bit are required to ensure a congruent set of local next states; these local states are voted to produce the next state in each ICIS. If the local next states in each ICIS produce a three-way disagreement (for a triplex FTP), the voted result could be the code for a non-existent state or a state for which the transition is incorrect. The voter detects a three-way disagreement and does not allow the next state latch to be updated. Similarly, disagreement in a duplex ICIS does not allow the change to the next state. The ICIS state exchange hardware is designed to conform to the same principles of fault tolerance that guided the implementation of the data exchange network used in the AIPS core FTP.

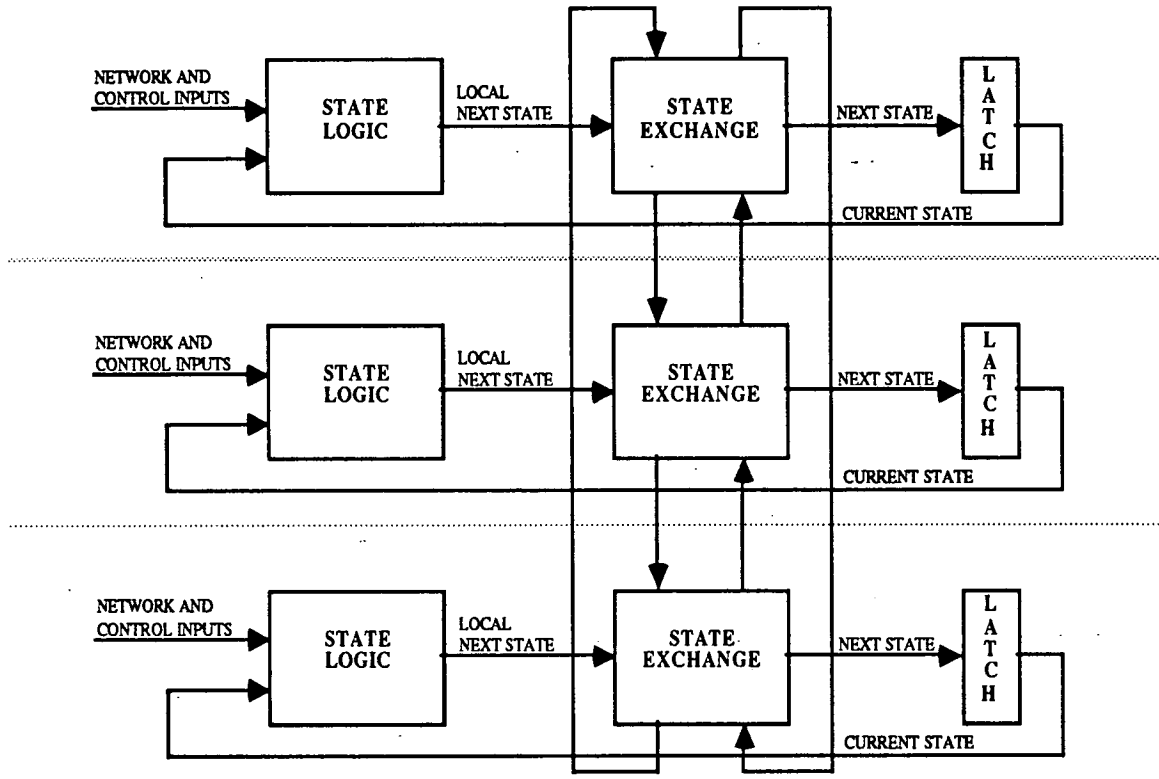


Figure 6-5. Next State Logic For Triplex ICIS

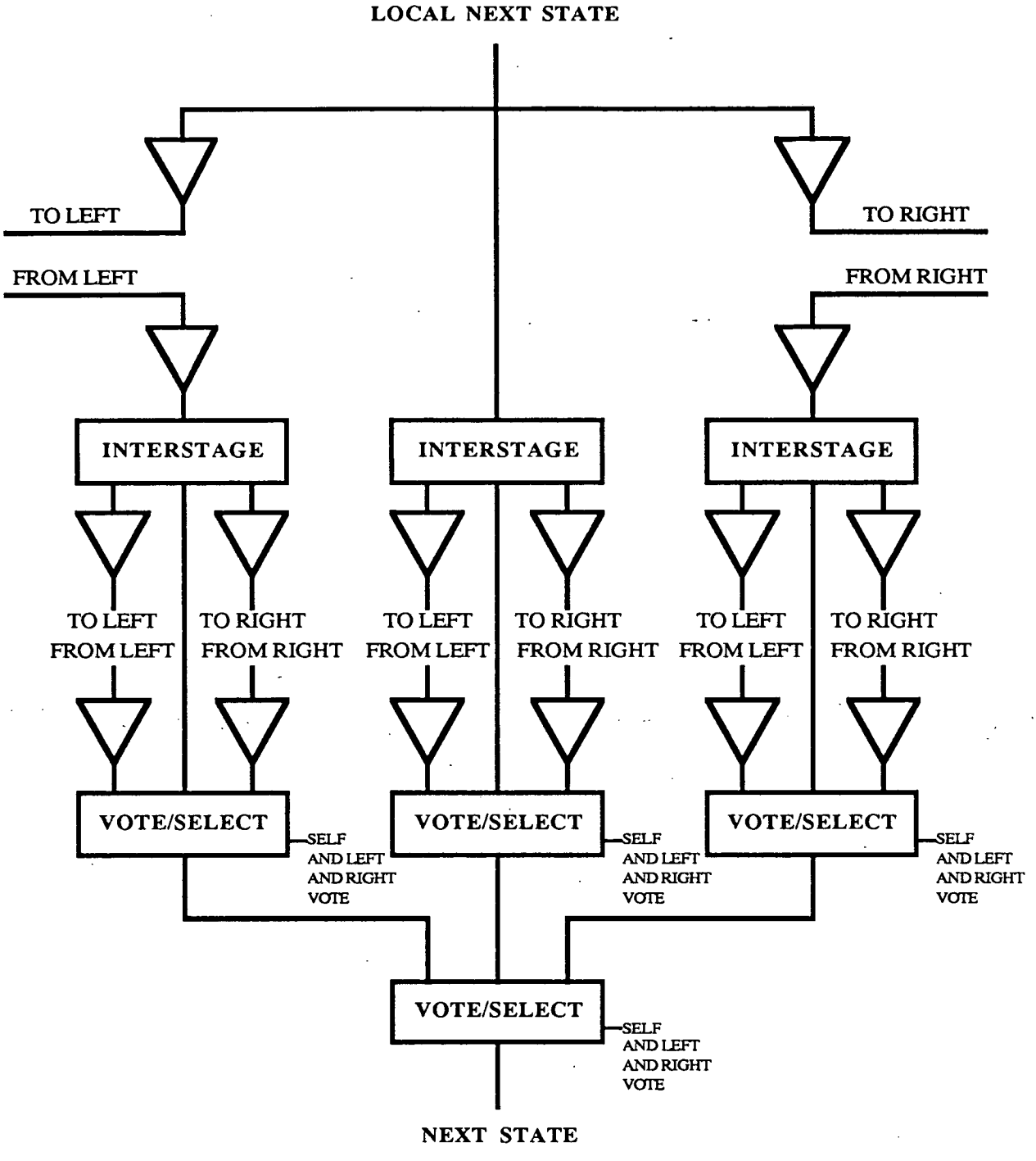


Figure 6-6. Congruent State Exchange Logic For One ICIS

6.2.3 ICIS Interface to the FTP

An important aspect of the AIPS FTP architecture is its ability to operate efficiently in a distributed computational environment. The architecture of the FTP interface to the inter-computer network has been designed from the very start to facilitate a congruent flow of data to/from other FTPs efficiently and with minimum software overheads and intervention. Figure 6-7 shows a schematic of the IC network interface hardware for one FTP channel. Each FTP channel, regardless of the host FTP redundancy level, listens to all three layers of the IC network. The ICIS interface electronics for a channel is divided into three logic modules, each of which interfaces with one IC node as illustrated in Figure 6-7. The ICIS modules interface with the intra-channel voter/selector logic, which is also called the LMN Voter/Selector and is resident on the shared bus controller card, via three 8-bit parallel backplane buses. The 24-bit address format for devices in the LMN region of the FTP address space is also shown in this figure. The two most significant bits in the CPU address space, when set to ' 1 0', select the LMN region. The next three bits select one of the devices on the LMN buses. Thus, up to eight devices can be accommodated per channel. Apart from ICIS, examples of other devices are I/O Sequencers (IOS) and Mass Memory Interface Sequencers (MMIS). The ICIS, IOS, and MMIS interface electronics appear as dual ported memory locations to the processor and the specific locations within a selected device are addressed by bits 0 to 11 and bit 15. (Note that no MMISes have been designed or implemented for the AIPS Distributed Engineering Model.)

This interface is very powerful, and it is tied directly into the inter-channel data exchange mechanism. Data can be routed from the interfaces through the intra-channel voter/selector logic and across the inter-channel voter/selector into the processor memory with a single processor move instruction. The documentation for the ICIS redundancy management software discusses the use of the intra-channel and inter-channel hardware for processing data received from other FTPs on the IC network in detail. Some of the possible data flows are discussed here.

Figure 6-8 shows the data flow from an ICIS to the CP through the LMN voter/selector. The LMN bits (16-18) in the address space specify the operation performed on the data by the voter/selector logic. Bits ABC (12-14) are set to '0' indicating that data is not routed through the cross-channel exchange hardware. If the LMN bits are '1 1 1', the LMN voter reads three locations specified by the 13 address bits marked 'a', performs a bit-for-bit majority vote on them and deposits the result in the CP memory address specified in the move instruction. Any disagreements are recorded in the LMN error register. Other combinations of LMN bits can be used to compare data on any two network layers such as L and M, M and N, or N and L. Data can also be selected from only one layer and passed through the voter/selector logic unchanged. Thus, it is possible to route data received from simplex, duplex, and triplex FTPs through the interface electronics and into the FTP very efficiently, while at the same time satisfying all the requirements for data congruency.

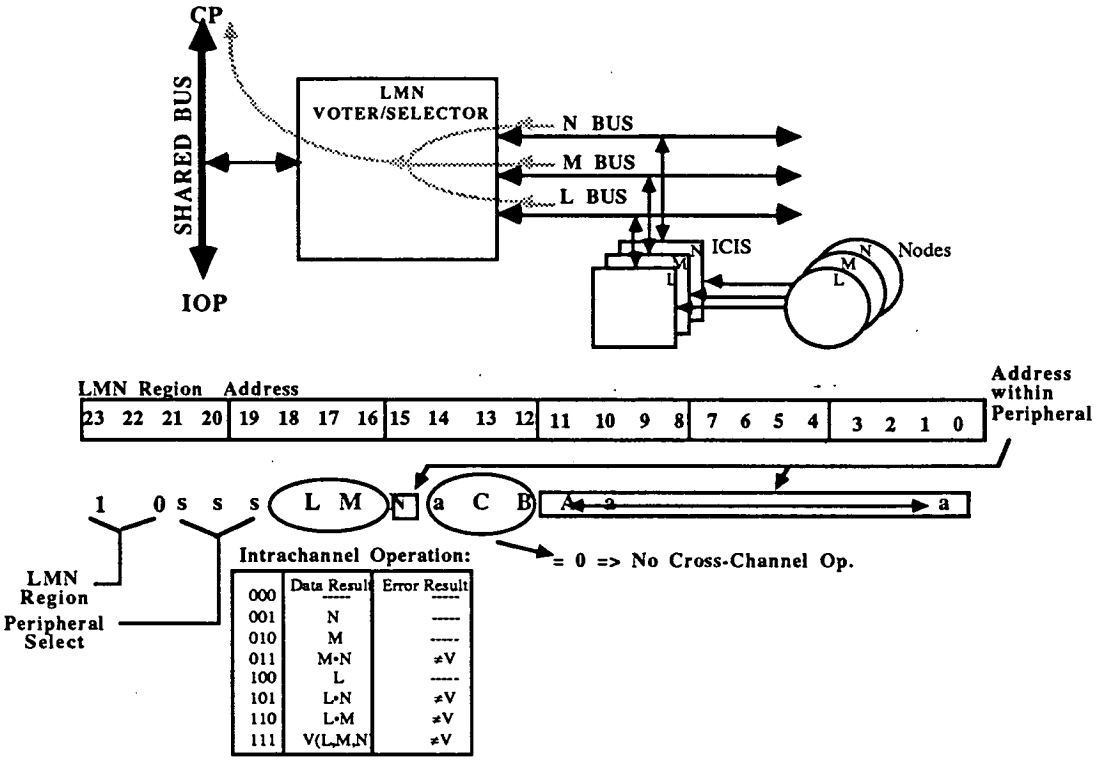


Figure 6-7. IC Network Interface for One FTP Channel

As stated previously, Figure 6-8 shows the data flow from an ICIS, through the LMN voter/selector, through the cross-channel data exchange and then into the CP. In this case, the ABC bits specify the operations to be performed by the cross-channel hardware. When the source of data is in the LMN region, any errors detected by the cross-channel hardware are recorded in an error register which is separate from the error register that is maintained for accesses in the non-LMN region. This separation of error information keeps the FDI related to the core FTP separate from the FDI associated with the IC hardware.

Figure 6-9 shows the data flow from a CP to the ICIS in one FTP channel. Each FTP channel is enabled to write on only one network layer. By choosing the correct bits in LMN and ABC fields, a triplex FTP can coordinate the channel activity so that it writes synchronously to all three network layers.

For more details concerning the design and implementation of the Inter-Computer Interface Sequencer, refer to Appendix C.

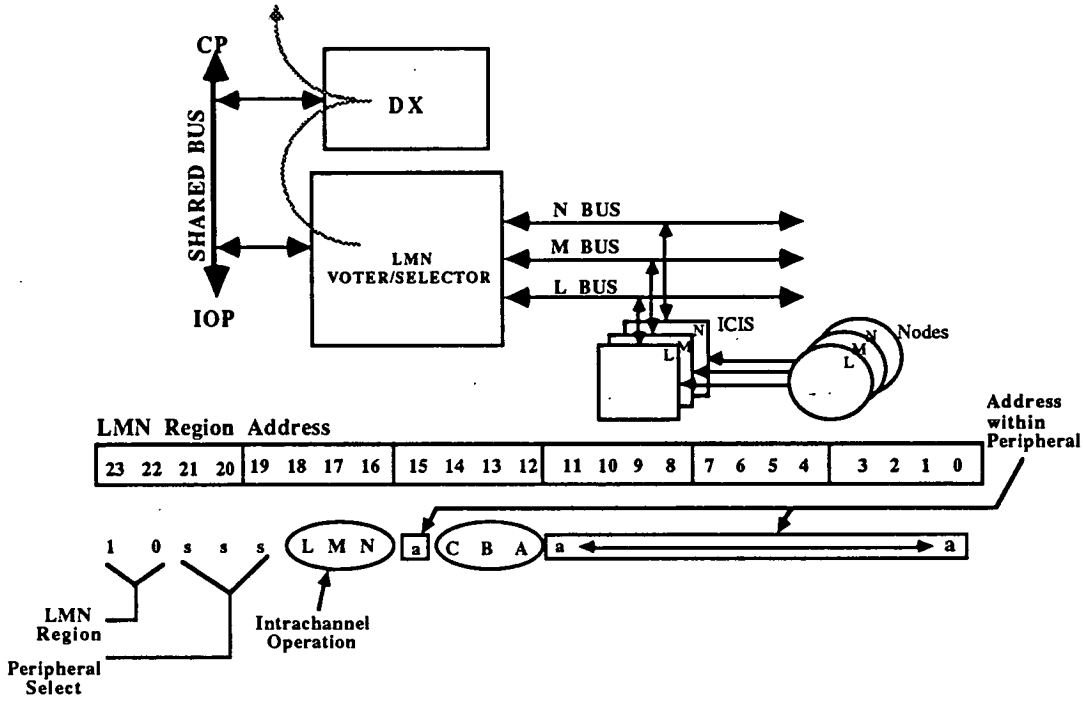


Figure 6-8. Data Flow through LMN and Cross-Channel Hardware

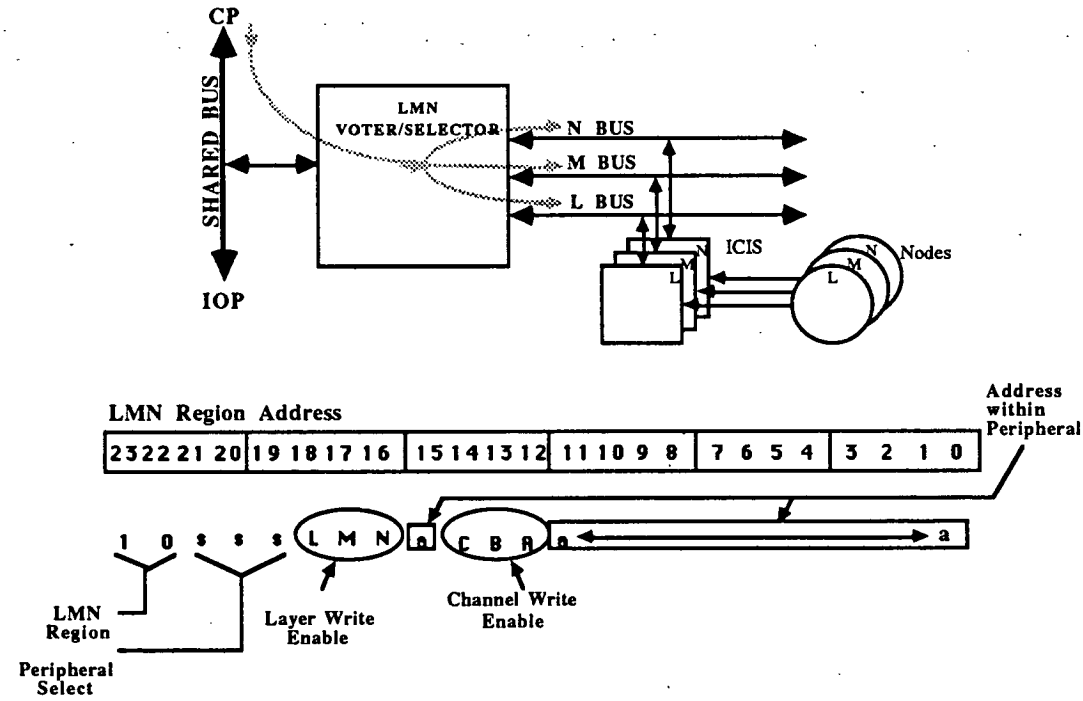


Figure 6-9. Data Flow from CP to ICIS

7.0 CONCLUSIONS AND RECOMMENDATIONS

Inter-Computer Communication Services for the distributed configuration of the AIPS engineering model have been designed, implemented and tested. This software provides the redundancy management and operating system support for the distributed communication between GPCs. The IC Network Manager, IC Layer Managers, and ICIS Redundancy Manager provide reliable communication over a triple layered Byzantine Resilient network which dynamically masks single faults. These functions are responsible for the initial growth of the IC network, the FDIR of the IC network and ICIS, and the source congruency of incoming messages. The Synchronous Communication Manager, User Services and the Message Send Receive functions provide local and distributed inter-function communication as a transparent service to the user. These functions support both synchronous and asynchronous communication in point to point mode and asynchronous communication in broadcast mode.

The Synchronous Communication Manager and the IC Network Growth with Contention functions of Inter-Computer Communication Services have been designed but have not been implemented and tested. With the exception of these two functions, all functions of the Inter-Computer Communication Services have been demonstrated.

The AIPS distributed engineering model software, composed of the Inter-Computer Communication Services, the Local System Services, and the Input/Output System Services, contains 98,681 lines of Ada source code, which includes 40,100 Ada statements and comments. When compiled and linked into two executable modules, the CP code (which includes instructions and global variables) requires 250,107 bytes of FTP memory and the IOP code requires 847,243 bytes.

7.1 Demonstration and Testing of Inter-Computer Communication Software

7.1.1 Demonstration Hardware and Software

In order to test the distributed AIPS engineering model under both fault and no fault conditions, additional hardware and software functions were designed and added to the system.

In order to test and demonstrate the ICIS redundancy manager, three fault injector boxes were built and attached to the ICIS cards of FTP 2. The fault injector boxes are able to inject a fault into an individual layer of the ICIS or the entire channel's ICIS. They are also able to inject ICIS link faults.

Interstage faults were injected using a switch that grounded the interstage's power supply. The fault injector previously used to test the FTMP was modified and used to inject pin level faults into the voting circuitry of FTP 3.

In order to demonstrate and test the ICCS in both broadcast and point to point communication between sites, a status broadcast task and pseudo Advanced Launch System (ALS) application tasks were implemented on the system.

7.1.2 Preliminary Testing of Inter-Computer Communication Services

The demonstration of the AIPS engineering model was first done under no fault conditions. The pseudo ALS application tasks execute on each site. Since each site has a Macintosh Apple as a front end display, the Macintosh monitor displays the changing variables that are involved in the point to point communication between sites. A display of the status of the entire AIPS system (four sites) is also displayed at each site. In order to test the broadcast mode, faults were injected at each site by turning off the power or disabling a channel's interstage. All faults were identified at all sites correctly. The pin level fault injector was also used to inject faults into site 3. Again the correct channel was identified at all sites.

The ICIS fault injector boxes were used to test the ICIS Redundancy Manager. Multiple faults were injected into the ICIS on site 2. All faults were correctly identified and inter site communication continued even with multiple faults in the ICIS.

The IC Network Manager was tested by injecting faults manually to various nodes and links of the IC Network by pulling links and resetting nodes. All faults were correctly identified and inter site communication continued even with multiple faults in the IC network.

Extensive systematic testing of the AIPS distributed engineering model for performance and reliability under fault-free and degraded conditions remains to be done. The hardware fault injector will be used for much of this testing.

7.1.3 Performance Metrics

Some performance metrics were gathered using both logic analyzers and the real-time clock to make time measurements. These metrics recorded for a sample IC communication are presented in Figure 7-1. Twenty samples were taken each time and the numbers were very consistent. The performance was measured from the time the source application task called the SEND_OUTPUT routine until the time the sink application task had a message available. The total time was 28.4 milliseconds. The following components of the AIPS distributed engineering model were used :

1) two triplex sites (FTP 2 and FTP 3) with 68010 processors with 7.9 MHz. clocks
2) custom IC Interface Sequencers (ICIS) with 7.9 MHz. clocks
3) (three) 2 MBit/second IC buses
4) 15 custom network nodes each with a 68701 processor and a 2MHz. clock
5) Verdix 5.5 compiler/RTS
6) Message length of 64 bytes

Location	Function	Overhead(Verdix 5.5)
Source FTP	SEND_OUTPUT (Add message to output queue, set event)	3.7 ms
Source FTP	Time between event and MSR	1.4 ms
Source FTP	Message Send Receive (MSR)	5.7 ms
IC Net	Time for ICIS and IC Network transmission	.4 ms
Sink FTP	Ave time between polling for msg	2.5 ms
Sink FTP	ICIS RM (make msg congruent, check for errors)	9.7 ms
Sink FTP	Time to do context switch	.9 ms
Sink FTP	MSR (Time between task start and when msg is available for user)	3.2 ms
Sink FTP	GET_INPUT (Remove message from buffer and pass to user)	.8 ms
	Total Time	**28.4 ms.**

Figure 7-1. IC Communication Latency

As indicated by Figure 7-1, the message latency on the network is about 28 milliseconds. The performance penalties are caused by the processor, compiler and data exchange hardware. These penalties are likely to be reduced significantly with advances in processor, compiler and hardware technologies. As discussed in the next section, the message latency for a state-of-the-art implementation will be reduced by a factor of 60 to less than 2 milliseconds. Thus, the system will easily meet the timing constraints of real-time flight control applications. For example, it will be practical for 100 Hz tasks executing on distributed processing sites to exchange data over the network at 100 Hz or even higher frequencies.

7.2 Future Work

The functions of the AIPS System Manager have not been designed or implemented. This work needs to be done in order to complete the distributed AIPS engineering model. There are also several areas of AIPS Inter-Computer Communications Services that remain to be addressed in the AIPS program. These include implementation of the Synchronous Communication Manager, the implementation of the IC Network Growth with Contention, and modifications that should be made to the ICCS Transport Layer for functional or performance improvement. Appendix D contains a list of the proposed modifications to the ICCS Transport Layer.

As discussed in the previous section, the processor used for the AIPS distributed engineering model was a Motorola 68010 operating at 7.9 MHz clock. The distributed inter-function communication time measured on this hardware and reported in the preceding section was 28.4 msec. Other benchmarks were measured on both the AIPS 68010 FTP and a 68020 FTP with a 14.7 MHz clock. The 68020 FTP outperformed the 68010 AIPS FTP by a factor of six [5]. It is projected that a state-of-the-art processor will outperform the 14.7 MHz 68020 by a factor of 10 [6]. Combining these numbers, a state-of-the-art processor with a mature Ada compiler should outperform the AIPS 68010 processor by a factor of 60. Therefore, it is predicted that distributed communication executing on state-of-the-art processors will easily meet the timing constraints of real time flight control applications. Empirical measurements of AIPS software executing on state-of-the-art hardware should be carried out to verify these performance predictions.

8.0 REFERENCES

1. L. Burkhardt, L. Alger, R. Whittredge, and P. Stasiowski, "Advanced Information Processing System: Local System Services", NASA Contractor Report 181767, April, 1989.

2. T. Masotto and L. Alger, "Advanced Information Processing System: Input/Output System Services", NASA Contractor Report 181874, August, 1989.

3. G. Nagle, L. Alger and A. Kemp, "Advanced Information Processing System: Input/Output Network Management Software", NASA Contractor Report 181678, May, 1988.

4. J. Martin, Distributed Processing Software and Network Strategy, Savant Research Studies, Lancashire, England, October 1979.

5. R. E. Harper, L. S. Alger, and J. H. Lala, "Advanced Information Processing System Design and Validation Knowledgebase", NASA Contractor Report 187544, September 1991.

6. R. Cole, "Advanced Information Processing System for Advanced Launch System: Hardware Technology Survey and Projections", NASA Contractor Report 187555, September 1991.

APPENDIX A: NODE SPECIFICATION

The input/output network is comprised of simplex nodes. These nodes are interconnected by links. A node is a communication switching point with five input/output ports. Figure A-1 is a basic representation of a node. The internal construction of each port of a node is shown in Figure A-2. Since a node does not have knowledge of the configuration of the network it must always have its receivers enabled. Reconfiguration commands can be accepted from any port whether enabled or not. Configuration commands enable selected ports. Ports are reconfigured whenever necessary and can be temporarily modified for single response frames. As a message is received on an enabled port it regenerates and retransmits the received data. At the same time, the message is decoded within the node. If the message is addressed to the node it responds to the command embedded within the data. If the message is addressed elsewhere it checks for a valid transmission, latches observed error conditions and resets the receiver for the next transmission.

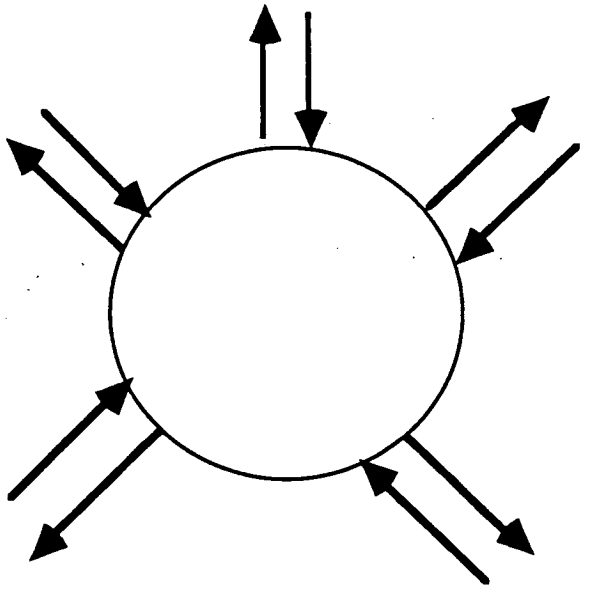


Figure A-1. AIPS NODE

Some components are unique to a port and some are shared by all the ports. Figure A-2 shows the basic construction of a node. The components within the dotted lines are unique to each port and are repeated five times. The components outside of the dotted lines form the node control section and are not repeated. The following is a description of the basic components of the node.

Port Components (unique to each port)

1. Receiver

   The receiver accepts the signal level on a link and converts it to the internal logic level of the port. The receiver also isolates the node from electrical failures of the link.

2. Protocol Decoder

   The protocol decoder accepts the serial data stream from the receiver and checks for protocol compliance and transmission induced errors. It then assembles the message into parallel words utilizing its clock extraction section. These parallel words are stored in a receive FIFO for the control sequencer to examine.

3. Clock Extractor

   Since the data transmission rate is 2 MHz, and all elements (FTPs, nodes, etc.) are operating on independent oscillators, it is necessary to generate a clock for the decoder. This clock is synchronized to the first edge of data that it sees, and it remains usable for the maximum message length.

4. Signal Regeneration Logic

   The signal regeneration logic is used to reconstruct the fidelity of the transmission. The passage of the signal through circuit elements in the node and the variability of the frequency of individual oscillators would degrade the signal if it were not reconstructed in each node. After several transitions through circuit elements the transmission could appear to be modified. The input to the regeneration logic is the OR of all the enabled port receivers and the protocol encoder output. The output of the regeneration logic is enabled or disabled by the port enable register and is applied to the input of the port transmitter.

5. Transmitter

   The transmitter converts the output of the regeneration logic into the signaling levels used on the links.

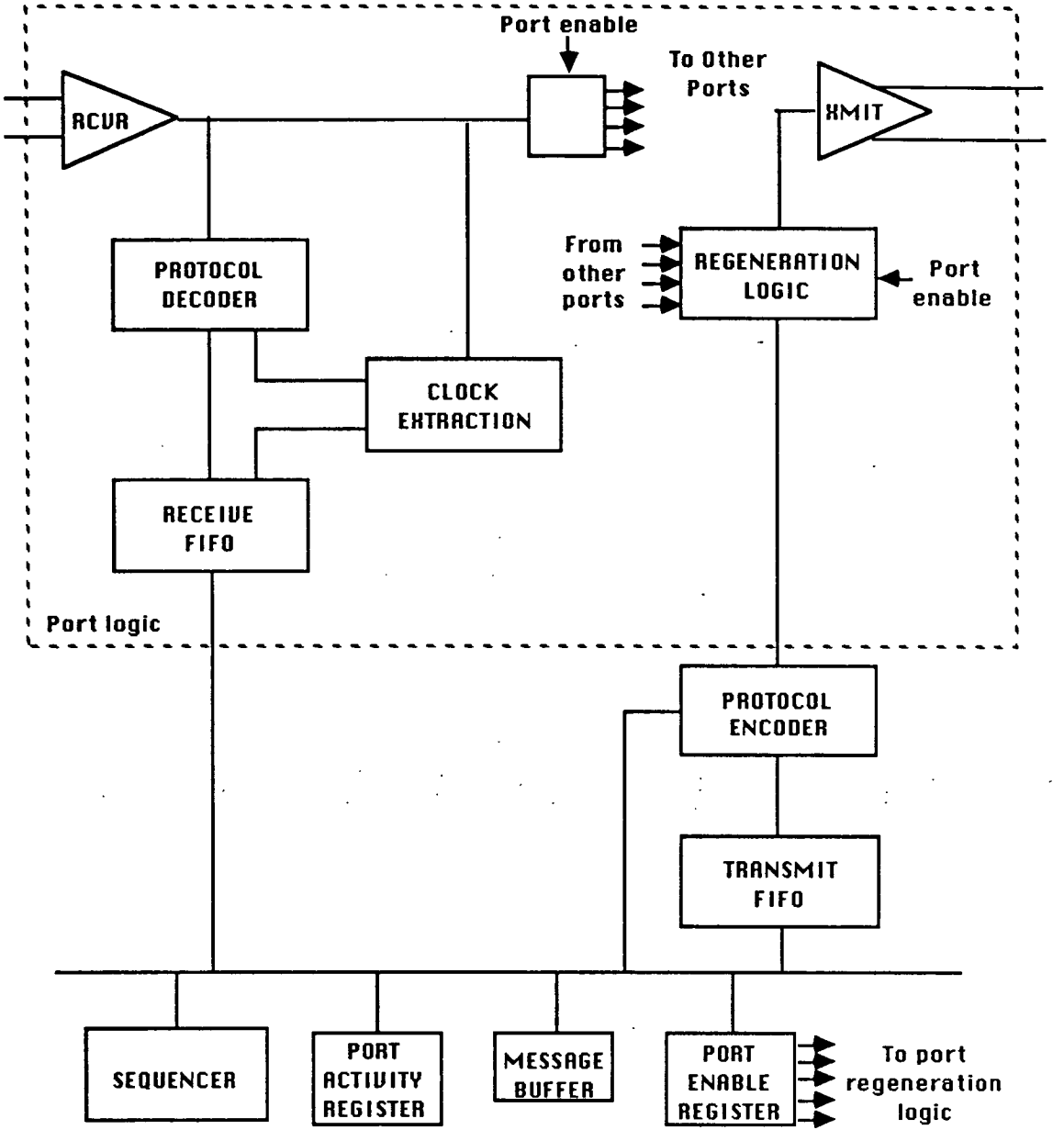


Figure A-2 NODE PORT

Control Components (shared by all the ports)

1. Node Sequencer and Control

   The node sequencer and control orchestrates the total operation of the node. It scans the port receive FIFOs for messages received from the links. If a message is found, it checks the address byte to determine if the message is addressed to this node. If the message is destined for this node, the sequencer then checks the bytes that follow the address byte to see if the rest of the message conforms to a proper node message. The message is acted upon only if it passes all tests. The sequencer is capable of reading the input FIFOs and writing to the transmitter FIFO, port enable register, and message buffer.

2. Port Enable Register

   The port enable register accepts the decoded commands from the sequencer and enables/disables the individual port reconstruction logic. The last command is stored until rewritten by the next command. The contents of this register are contained within the status message from the node.

3. Message Buffer

   The message buffer is a 64 byte long RAM which can be written into by an appropriate node command. The contents of this RAM can be returned by the node in place of a status message.

4. Port Activity Register

   The port activity register is set whenever a transition is detected on the port receiver.

5. Transmit FIFO

   The transmit FIFO holds the node response message for application to the protocol encoder.

6. Protocol Encoder

   The protocol encoder receives the node responses and encodes them into the link protocol. The output of the encoder is sent to the reconstruction logic of all ports.

Input Frame Message Format

The following is the format of an input frame sent to a node

1. Opening Flag
2. Node Address
3. Encoded Node Address
4. Operation Code
5. Port Enables and Control
6. Message Sum Check
7. Residue Bits
8. FCS
9. FCS
10. Closing Flag

Bit assignments within the transaction are as follows.

bit	7	6	5	4	3	2	1	0
Opening Flag	0	1	1	1	1	1	1	0
Node Address	Node Address Bits							
Encoded Address	Encoded Node Address Bits							
Op Code	Mode	Mode	Mode	Stat MsgB	Err	Res	Res	Res
Port Enable	Chg Port	Enb Once	Clr Stat	E	D	C	B	A
Sum Check	Sum Check Bits							
Residue Bits	Residue Bits							
FCS	FCS High Byte							
FCS	FCS Low Byte							
Closing Flag	0	1	1	1	1	1	1	0

1. OPENING FLAG: As defined in the HDLC specification, this flag is used to synchronize and separate transmissions.

2. NODE ADDRESS: The address of the node to which this message is directed.

3. ENCODED NODE ADDRESS: The encoded address of the node to which this transaction is directed. It has been placed in the byte that HDLC has defined as control. Since control code definition is defined by the user, in AIPS it is used as the encoded address to help shorten the response time and is the one's complement of the node address.

4. OPERATION CODE: Contains the code for the function to be performed by the addressed node. The following is the definition of those functions.

Bit 7	6	5	4	3	
1	1	1	R	E	Modify Port Enable Register as specified in next byte.
1	1	0	R	E	Reserved
1	0	1	R	E	Next byte to count register
1	0	0	R	E	Next byte to Address Reg H
0	1	1	R	E	Next byte to Address Reg L
0	1	0	R	E	Next byte to address specified by Address Register
0	0	1	R	E	Next byte to address specified by Address Register then +1 to Address Register
0	0	0	R	E	No modification to Port Enable Register (next byte ignored).

All valid input frames result in a response frame from the node. The content of the response frame is determined by the state of bit 4 as defined below.

Bit 4 R=1 Respond from Status Register
R=0 Respond from Message Buffer

The node can be commanded to send a response frame that contains a transmission error for testing purposes. This faulty frame can occur in conjunction with any of the above defined modes. A faulty frame is one in which the transmission is truncated, i.e. aborted. The choice of valid or faulty frames is determined by the state of bit 3 as defined below.

Bit 3 E=1 Respond with faulty Message
E=0 Respond with valid message

Modes 1, 2, 3, 4 and 5 are for specifying the parameters used to generate responses from the message buffer. If a response is specified from the message buffer, the node will respond with the number of bytes specified by the counter starting at the address contained in the Address Register. The contents of the counter and Address Register

are not changed by a response request. The counter and Address Register are modified as specified above using modes 3, 4 and 5. Modes 1 and 2 are used to load specified memory locations within the node. Mode 1 automatically increments the address register after each byte is stored at the present location specified by the address register. The Address register can only specify locations from 00C0 (Hexadecimal) to 00FF (Hex), a total of 64 bytes. Mode 2 is used to specify memory locations in a random access mode. Bits 2, 1, and 0 specify, in binary, the number of residue bits to be generated in a response frame.

5. PORT ENABLES AND CONTROL: If mode 7 is specified in the opcode byte, this byte is loaded into the port enable register. If bit 7 of this byte is set (=1), then the port enable register is changed permanently. However, if bit 7 is not set and bit 6 is set, the contents of the port enable register are modified for this transmission only. At the completion of this transmission the previous contents are reloaded into the port enable register. If both bits 7 and 6 are set at the same time, the node will respond as if only bit 7 were set, i.e. the port enable register will be permanently modified. Bit 5, if set, specifies that all status registers are to be cleared after this response is completed.

6. MESSAGE SUM CHECK: The contents of this byte are calculated such that a modulo 256 add of the Address byte, Encoded Address byte, OpCode byte, Port Enable byte, and this byte yield a result of zero. It is computed at the source and verified in the node to check for errors outside the transmission medium.

7. RESIDUE BITS: Used to differentiate node messages from all other transactions. There are three residue bits in a node message and the content of these bits is not specified.

8. FCS: This byte contains the high byte of the FCS as calculated in the transmitter.

9. FCS: This byte contains the low byte of the FCS as calculated in the transmitter.

10. CLOSING FLAG: This byte is defined by HDLC as the transmission terminator or separator.

Output Frame Message Format

The node always responds after a valid input frame. The output frame can be generated from: (A) the status register or (B) the message buffer.

A. Output Frame From the Status Register

When an output frame is requested from the status register it will take the following form.

1. Opening Flag
2. Node Address
3. Port Activity Seen
4. Transmission Errors Seen
5. Valid Frame Seen
6. Error in Node Messages Seen
7. Node Valid Frame Seen
8. Node Port Configuration
9. Sum Check
10. Residue Bits
11. FCS
12. FCS
13. Closing Flag

Bit assignments within the Output Frame from the status register are as follows.

bit	7	6	5	4	3	2	1	0
Opening Flag	0	1	1	1	1	1	1	0
Node Address	Node Address Bits							
Activity Seen	X	X	X	E	D	C	B	A
Transmission Errs	X	X	X	E	D	C	B	A
Valid Frame Seen	X	X	X	E	D	C	B	A
Node Error Seen	X	X	X	E	D	C	B	A
Node Valid Frame	X	X	X	E	D	C	B	A
Node Port Config	X	X	X	E	D	C	B	A
Sum Check	Sum Check Bits							
Residue	Residue Bits							
FCS	FCS High Byte							
FCS	FCS Low Byte							
Closing Flag	0	1	1	1	1	1	1	0

X=Reserved

1. OPENING FLAG: As defined in the HDLC specification, this flag is used to synchronize and separate transmissions.

2. NODE ADDRESS: The address of this node.

3. ACTIVITY SEEN: Whenever a transition on a link is detected at a port, whether enabled or not, the corresponding bit in the byte is set to a 1. These bits remain set until a clear status command is received in a valid input frame.

4. TRANSMISSION ERRORS: Whenever a node detects a transmission error, a bit is set for the corresponding port. These bits remain set until a clear status command is received in a valid input frame.

5. VALID FRAME SEEN: Whenever a frame is seen without transmission errors, the corresponding port bit is set. These bits remain set until a clear status command is received in a valid input frame.

6. NODE ERRORS SEEN: Whenever a frame is received addressed to this node and without transmission errors but not honored by this node, the bit corresponding to the port on which it was received is set. These bits remain set until a clear status command is received in a valid input frame.

7. NODE VALID FRAME: Whenever a node responds to an input frame the corresponding port bit in this byte is set. This bit is set before a response transmission and cleared after the response transmission if a clear status command is received.

8. NODE PORT CONFIGURATION: This byte is normally set to the present state of the port enable register. However, if the input transmission had requested a change of port configuration for this transmission only (ENB ONCE bit set), then the byte is set to the state to which the node will revert after this transmission.

9. MESSAGE SUM CHECK: The contents of this byte are calculated such that a modulo 256 add of the Address byte, Activity Seen byte, Transmission Errors byte, Valid Frame Seen byte, Node Errors Seen byte, Node Valid Frame byte, Node Port Configuration byte and this byte yield a result of zero. It is computed by the node to enable user detection.

10. FCS: The FCS bytes are a cyclic redundancy calculation performed by the HDLC transmitter and appended to the end of the frame.

11. CLOSING FLAG: The closing flag is the frame terminator.

B. Output Frame From the Message Buffer

An output frame from the message buffer is intended to be used as a test tool. The output frame information field contains the number of bytes specified in the counter starting at the address in the Address Register. The counter and Address Register must have been initialized prior to a request. The values in these registers remain unchanged until they are rewritten. A byte count of zero will result in 256 bytes being transmitted. The output frame will take the following form.

1. Opening Flag

2. Contents of Address specified by the Address Register

3. Contents of Address specified by the Address Register + 1
4. •
5. •
6. Contents of Address Specified by the Address Register + Counter
7. Residue bits
8. FCS
9. FCS
10. Closing Flag

APPENDIX B: TRANSPORT LAYER USER'S GUIDE

This appendix describes the interfaces to the Transport Layer available to a user who wants to send and receive messages. As described in Section 4.1.1, such a user expects to transmit and accept messages but without having any Session Layer functions performed for it (perhaps it *is* the Session Layer).

B.1. User Identification

All users of the Transport Layer must identify themselves, both to other users and to the Transport Layer. This can be done by updating one of two packages: ICCS_SYSTEM_USER_IDS (for system users) or ICCS_APPLIC_USER_IDS (for application uers). Listings of these packages are shown in Figures B-1 and B-2. These packages have defined a fixed number of user IDs, some with names such as "UNUSED" or "RESERVED" to indicate IDs planned for but not yet actually used. To enter a new user ID, a descriptive name must be substituted for one of the unused names. This name must be entered in two places: in the **type ic_user_id is . . .** statement and in the **for ic_user_id use . . .** statement. Note that only one of the two packages (system user IDs or application user IDs) should be updated for any given user.

For example, to add a DATA_STORAGE application as a new user, DATA_STORAGE would replace APPL_UNUSED_7 in both the **type ic_user_id is . . .** statement and in the **for ic_user_id use . . .** statement in the ICCS_APPLIC_USER_IDS package. Then the ICCS_APPLIC_USER_IDS package and the ICCS_USER_SERVICES_APP package must be recompiled, as well as any other application tasks which reference these two packages.

Similarly, to add a RESOURCE_ALLOCATOR task as a new system user, RESOURCE_ ALLOCATOR would replace RESERVED_13 in the **type ic_user_id is . . .** statement and the **for ic_user_id use ...** statement in the ICCS_SYSTEM_USER_IDS package. Then the ICCS_SYSTEM_USER_IDS package and the ICCS_USER_SERVICES_SYS package must be recompiled, as well as any other system tasks which use the IC communication services and therefore reference these two packages.

```
------------------------------------------------------------------------------
--                      Package Spec ICCS_SYSTEM_USER_IDS
--  Contains ids for all users of IC Communication Services.  These are
--  kept in a separate package so that new users can be added without
--  having to recompile the IC software.
--
--   Name          Date           Description
--
--  L. Burkhardt   20-Oct-1988    Created
------------------------------------------------------------------------------

package ICCS_SYSTEM_USER_IDS is

                                                  -- user ids
type ic_user_id is (NON_ICCS_USER,
                    ------------------
                    CP_STATUS_REPORTER,       -- System users
                    IOP_STATUS_REPORTER,
                    CP_RENDEZVOUS_MGR,
                    IOP_RENDEZVOUS_MGR,
                    CP_GLOBALDATA_MGR,
                    IOP_GLOBALDATA_MGR,
                    CP_TEST_APPL,
                    IOP_TEST_APPL,
                    CP_ST_BROADCAST,
                    IOP_ST_BROADCAST,
                    LOCAL_ICIS_MGR,
                    IC_NETWORK_MGR,
                    RESERVED_13,
                    RESERVED_14,
                    RESERVED_15,
                    ---------------
                    APPL_UNUSED_1,            -- Application users
                    APPL_UNUSED_2,
                    APPL_UNUSED_3,
                    APPL_UNUSED_4,
                    APPL_UNUSED_5,
                    APPL_UNUSED_6,
                    APPL_UNUSED_7,
                    APPL_UNUSED_8,
                    APPL_UNUSED_9,
                    APPL_UNUSED_10,
                    APPL_UNUSED_11,
                    APPL_UNUSED_12,
                    APPL_UNUSED_13,
                    APPL_UNUSED_14,
                    APPL_UNUSED_15,
                    APPL_UNUSED_16,
                    APPL_UNUSED_17,
                    APPL_UNUSED_18,
                    APPL_UNUSED_19,
                    APPL_UNUSED_20);

for  ic_user_id'size use 8;
```

Figure B-1. ICCS_SYSTEM_USER_IDS Package

```
for ic_user_id use
                    (NON_ICCS_USER        => 0,
                     -------------------------
                     CP_STATUS_REPORTER   => 1,
                     IOP_STATUS_REPORTER  => 2,
                     CP_RENDEZVOUS_MGR    => 3,
                     IOP_RENDEZVOUS_MGR   => 4,
                     CP_GLOBALDATA_MGR    => 5,
                     IOP_GLOBALDATA_MGR   => 6,
                     CP_TEST_APPL         => 7,
                     IOP_TEST_APPL        => 8,
                     CP_ST_BROADCAST      => 9,
                     IOP_ST_BROADCAST     => 10,
                     LOCAL_ICIS_MGR       => 11,
                     IC_NETWORK_MGR       => 12,
                     RESERVED_13      => 13,
                     RESERVED_14      => 14,
                     RESERVED_15      => 15,
                     -------------------------
                     APPL_UNUSED_1      => 16,
                     APPL_UNUSED_2      => 17,
                     APPL_UNUSED_3      => 18,
                     APPL_UNUSED_4      => 19,
                     APPL_UNUSED_5      => 20,
                     APPL_UNUSED_6      => 21,
                     APPL_UNUSED_7      => 22,
                     APPL_UNUSED_8      => 23,
                     APPL_UNUSED_9      => 24,
                     APPL_UNUSED_10     => 25,
                     APPL_UNUSED_11     => 26,
                     APPL_UNUSED_12     => 27,
                     APPL_UNUSED_13     => 28,
                     APPL_UNUSED_14     => 29,
                     APPL_UNUSED_15     => 30,
                     APPL_UNUSED_16     => 31,
                     APPL_UNUSED_17     => 32,
                     APPL_UNUSED_18     => 33,
                     APPL_UNUSED_19     => 34,
                     APPL_UNUSED_20     => 35 );

end ICCS_SYSTEM_USER_IDS;
```

Figure B-1. ICCS_SYSTEM_USER_IDS Package (cont.)

```
--------------------------------------------------------------------------
--                    Package Spec ICCS_APPLIC_USER_IDS
--  Contains ids for all users of IC Communication Services.  These are
--  kept in a separate package so that new users can be added without
--  having to recompile the IC software.
--
--   Name          Date          Description
--
--  L. Burkhardt   20-Oct-1988   Created
--------------------------------------------------------------------------

package ICCS_APPLIC_USER_IDS is

                                                -- user ids
type ic_user_id is (NON_ICCS_USER,
                    ------------------
                    RESERVED_1,                 -- System users
                    RESERVED_2,
                    RESERVED_3,
                    RESERVED_4,
                    RESERVED_5,
                    RESERVED_6,
                    RESERVED_7,
                    RESERVED_8,
                    RESERVED_9,
                    RESERVED_10,
                    RESERVED_11,
                    RESERVED_12,
                    RESERVED_13,
                    RESERVED_14,
                    RESERVED_15,
                    --------------------
                    CCP,                    -- Application users
                    RANGE_SAFETY,
                    PROP_CONTROL,
                    SENSOR_PROC,
                    WIND_DETERMIN,
                    COMMAND_AND_TELEM,
                    APPL_UNUSED_7,
                    APPL_UNUSED_8,
                    APPL_UNUSED_9,
                    APPL_UNUSED_10,
                    APPL_UNUSED_11,
                    APPL_UNUSED_12,
                    APPL_UNUSED_13,
                    APPL_UNUSED_14,
                    APPL_UNUSED_15,
                    APPL_UNUSED_16,
                    APPL_UNUSED_17,
                    APPL_UNUSED_18,
                    APPL_UNUSED_19,
                    APPL_UNUSED_20);

for  ic_user_id'size use 8;
```

Figure B-2. ICCS_APPLIC_USER_IDS Package

```
for ic_user_id use
                    (NON_ICCS_USER      => 0,
                     ----------------------------
                     RESERVED_1         => 1,
                     RESERVED_2         => 2,
                     RESERVED_3         => 3,
                     RESERVED_4         => 4,
                     RESERVED_5         => 5,
                     RESERVED_6         => 6,
                     RESERVED_7         => 7,
                     RESERVED_8         => 8,
                     RESERVED_9         => 9,
                     RESERVED_10        => 10,
                     RESERVED_11        => 11,
                     RESERVED_12        => 12,
                     RESERVED_13        => 13,
                     RESERVED_14        => 14,
                     RESERVED_15        => 15,
                     ----------------------------
                     CCP                => 16,
                     RANGE_SAFETY       => 17,
                     PROP_CONTROL       => 18,
                     SENSOR_PROC        => 19,
                     WIND_DETERMIN      => 20,
                     COMMAND_AND_TELEM  => 21,
                     APPL_UNUSED_7      => 22,
                     APPL_UNUSED_8      => 23,
                     APPL_UNUSED_9      => 24,
                     APPL_UNUSED_10     => 25,
                     APPL_UNUSED_11     => 26,
                     APPL_UNUSED_12     => 27,
                     APPL_UNUSED_13     => 28,
                     APPL_UNUSED_14     => 29,
                     APPL_UNUSED_15     => 30,
                     APPL_UNUSED_16     => 31,
                     APPL_UNUSED_17     => 32,
                     APPL_UNUSED_18     => 33,
                     APPL_UNUSED_19     => 34,
                     APPL_UNUSED_20     => 35 );

end ICCS_APPLIC_USER_IDS;
```

Figure B-2. ICCS_APPLIC_USER_IDS Package (cont.)

A maximum of 35 user IDs (15 system, 20 application) have been provided. If either subset exceeds its maximum, both ICCS_SYSTEM_USER_IDS and ICCS_APPLIC_USER_IDS packages must be modified to reflect the new maximum. Then both of these packages and the system tasks and application tasks which reference them must be recompiled.

If the total number of user IDs exceeds 41, the ICCS_DATA_TYPES package must be modified, specifically, the **user_t** type. Then all packages dependent directly or indirectly on this package must be recompiled. The ICCS_DATA_TYPES package is shown in Figure B-3. The recompilation order for all dependent programs is shown in Figure B-4.

```
-------------------------------------------------------------------------
--                    Package Spec ICCS_DATA_TYPES
--  Contains data types and high-level variables related to IC communication
--
--
--   Name          Date           Description
--
--  L. Burkhardt   23-May-1988    Created
-------------------------------------------------------------------------
with SYSTEM, LSS_ON_CARD_RAM, UNCHECKED_CONVERSION;

with ICCS_USER_DATA_TYPES, LSS_MEMORY;
use  ICCS_USER_DATA_TYPES, LSS_MEMORY;


package ICCS_DATA_TYPES is


type msec_t is range -2**31..2**31 - 1;      -- from RTS timer_support package
for msec_t'size use 32;

type user_t is range 0..40;                  -- my definition of user_id
for user_t'size use 8;                       -- should match what's in System_User_Ids
                                             -- and Applic_User_Ids



--  Identify this GPC

THIS_GPC : gpc_t;


end ICCS_DATA_TYPES;
```

Figure B-3. ICCS_DATA_TYPES Package

```
-------------------------------------------------------------------------
---  RECOMPILATION ORDER when ICCS_DATA_TYPES package has been modified
-------------------------------------------------------------------------

---  ICCS Packages

ICCS_DATA_TYPES
ICCS_ICIS_TYPES

ICCS_CP_IOP_COMMON
ICCS_CP_IOP_COMMON_B
ICCS_USER_SERVICES
ICCS_USER_SERVICES_SYS
ICCS_USER_SERVICES_APP
ICCS_ERROR_LOG

ICCS_MESSAGE_SEND_RCV

ICCS_RM_OBJS
ICIS_LOCAL_MANAGER

ICCS_MESSAGE_SEND_RCV_B
ICCS_MESSAGE_SEND_RCV_S1
ICCS_MESSAGE_SEND_RCV_S2
ICCS_MESSAGE_SEND_RCV_S3

ICCS_USER_SERVICES_B
ICCS_USER_SERVICES_SYS_B
ICCS_USER_SERVICES_APP_B

ICCS_ICIS_INIT
ICCS_ERROR_LOG_B


--- ICIS RM Packages

ICCS_RM_UTIL
ICCS_RM_PACKET

ICCS_RM_CONG_DATA
ICCS_RM_ACTIVE_LAYER
ICCS_RM_BC_ANALYSIS
ICCS_RM_SDLC

ICCS_RM_UTIL_B
ICCS_RM_PACKET_B

ICCS_RM

ICCS_RM_B
ICCS_RM_CONG_DATA_B

ICIS_LOCAL_MANAGER_B
```

Figure B-4. Recompilation Order for Packages Dependent on ICCS_DATA_TYPES

```
---  NETWORK MGR FILES dependent on ICCS packages

ICCS_NET_MGR_B
ICCS_NET_MGR_CONFIG_B
ICCS_NET_MGR_COLLECT_B
ICCS_NET_MGR_MISC_B
ICCS_NET_SELF_TEST_B


---  GPC FDIR Packages dependent on ICCS packages

LSS_CONFIG_B
ICCS_FDIR_TIME_CP
ICCS_FDIR_TIME_IOP
ICCS_FDIR_TIME_CP_B
ICCS_FDIR_TIME_IOP_B
ICCS_TFDI_IOP_B
ICCS_TFDI_CP_B
LSS_SYNC_B


---  DISPLAY Packages dependent on ICCS packages

ICCS_MAC_IO.A
ICCS_MAC_IO_B.A
ICCS_MAC_DISP.A
ICCS_MAC_DISP_B.A
ICCS_MAC_APPL_DISP
ICCS_MAC_APPL_DISP_B

ICCS_MAC_DISP_MAIN_CP
ICCS_MAC_DISP_MAIN_CP_B
ICCS_MAC_DISP_MAIN_IOP
ICCS_MAC_DISP_MAIN_IOP_B
ICCS_DISP_MAIN_CP
ICCS_DISP_MAIN_CP_B
ICCS_DISP_MAIN_IOP
ICCS_DISP_MAIN_IOP_B


---  CP AND IOP MAIN PROGRAMS

ICCS_MAIN_PROG_CP
ICCS_MAIN_PROG_IOP
```

Figure B-4. Recompilation Order for Packages Dependent on ICCS_DATA_TYPES (cont.)

B.2 Interface Routines

This section describes the interface routines available to a Transport Layer user.

Figure B-5 lists the available routines. A brief description and the name of the defining package is given for each routine.

Figure B-6 describes the parameters required for each routine. In addition, each routine is identified as being either a function (F) or a procedure (P).

Figure B-7 lists the packages that define the required data types.

ROUTINE	PACKAGE	DESCRIPTION
IDENTIFY_TASK_LOCATION	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables a user to enter its initial location into a central database so that the user's GPC, processor id and Task Control Block are known throughout the system. All Transport Layer users must invoke this routine during task initialization.
OUTPUT_SETUP	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables user to allocate buffers which will be used as temporary storage areas for messages the user wants to send. All users wishing to send output messages must invoke this routine during task initialization.
INPUT_SETUP	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables user to allocate buffers which will be used as temporary storage areas for input messages that arrive for the user. All users expecting to receive messages must invoke this routine during task initialization.
SEND_OUTPUT	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables user to send a message.
CHECK_OUTPUT_STATUS	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables user to check on the status of an output message. Such a check is optional and is most useful for one-time (as compared to periodic messages).
GET_INPUT	ICCS_USER_SERVICES_SYS, ICCS_USER_SERVICES_APP	Enables user to poll for input messages and to receive them.

Figure B-5. Interface Routines for Transport Layer Users

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
IDENTIFY_TASK_LOCATION (F)	Self identification	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Own GPC	**gpc_t**	IN	GPCs (or combinations) in the system. Values are: ALL_OTHER_GPCS - All GPCs in the system except this one SITE1 - GPC 1 SITE2 - GPC 2 SITE3 - GPC 3 SITE4 - GPC 4 NO_GPC - Null value ALL_GPCS - All GPCS in the system including this one TASK_DEPENDENT - The GPC given in the Task Location Table for the relevant task NOTE: When a task is specifying its location, the value used must be SITE1 - SITE4 or ALL_GPCS. The other values for this type are used to specify the destination GPC during message transmission.
	Own processor	**processor_id**	IN	Processors in the system. Values are: CP - Computational Processor IOP - Input/Output Processor
	Own task id	**task_id**	IN	Address of the Task Control Block for this user
	Dummy return value	**boolean**	OUT	Allows the routine to be invoked in a package specification. The value itself has no meaning.

Figure B-6. Parameters Required for the Interface Routines

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
OUTPUT_SETUP (P)	Self identification	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Number of buffers required	**Integer**	IN	This number should reflect the number of output messages the user expects to have in progress at any one time.
	Buffer size	**Integer**	IN	Buffer size in bytes.
INPUT_SETUP (P)	Self identification	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Number of buffers required	**Integer**	IN	Number of buffers to be allocated. Should reflect the number of input messages the user expects to be receiving at any one time.
	Buffer size	**Integer**	IN	Buffer size in bytes.
	User notification flag	**boolean**	IN	TRUE - User should be signaled with an event when input arrives FALSE - User should not be signaled with an event.
	User event	**a_Event**	OUT	Event the task must use to schedule itself, if previous parameter is TRUE.

Figure B-6. Parameters Required for the Interface Routines (cont.)

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
SEND_OUTPUT (P)	Self identification	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Message starting address	**address**	IN	Address of the message to be sent. Obtained by using the 'address attribute.
	Destination GPC	**gpc_t**	IN	Refer to description of gpc_t given previously for IDENTIFY_TASK_LOCATION routine.
	Destination user	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Message length	**integer**	IN	Length of the message in bytes
	Message priority	**msg_priority_t**	IN	A user-assigned priority which Ranges from 1 (lowest priority) to 100 (highest priority). Not used by Transport Layer.
	User-defined message id	**short_integer**	IN	This parameter is used by a sender who wants to check later on the status of the message. A sender who will not be making this check may specify any value here.
	User status-check flag	**boolean**	IN	TRUE - Sender will check later on status of message transmission FALSE - Sender will not check later.

Figure B-6. Parameters Required for the Interface Routines (cont.)

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
SEND_OUTPUT (cont.)	Pre-transmission error flag	**output_error_t**	OUT	A code identifying errors in message syntax or a known situation that will result in unsuccessful transmission. NO_ERRORS - Message passed all preliminary checks INVALID_BYTE_COUNT - The message size was less than 1 or was bigger than the user's allocated holding buffers. INVALID_GPC_ID - An invalid code was specified for the destination GPC. INVALID_USER_ID - An invalid code was specified for the self identification. NO_FREE_OUTPUT_BUFFER - There is no room to temporarily store the message until it can be sent. DEST_GPC_NOT_STARTED - The destination GPC is currently not operational. NETWORK_NOT_IN_SERVICE - The IC network is currently not operational DEST_TASK_UNKNOWN - An invalid code was specified for the destination task.

Figure B-6. Parameters Required for the Interface Routines (cont.)

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
CHECK_OUTPUT_STATUS (F)	Self identification	**lc_user_id**	IN	Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	User-defined message id	**short_Integer**	IN	This should be the same id previously passed to the SEND_OUTPUT routine when sending the message in question.
	Message status (return value)	**output_status_t**	OUT	Current status of the transmission. Values are: NO_ERRORS - Message has been successfully transmitted MSG_ID_NOT_FOUND - History about a message with the specified message id could not be found. Either the user waited too long before checking on the status or used an incorrect message id. MSG_NOT_COMPLETE - The message is in the process of being sent. TRANSMIT_ERROR - A hardware failure occurred during message transmission. DEST_GPC_NOT_RESPONDING - The destination GPC is not acknowledging receipt of the message DEST_GPC_CANT_DELIV - The message was received by the destination GPC, but no space was available to store the message for the destination user.

Figure B-6. Parameters Required for the Interface Routines (cont.)

ROUTINE	PARAMETERS			
	DESCRIPTION	TYPE	IN/OUT	EXPLANATION
GET_INPUT (P)	Self identification	lc_user_id	IN	Refer to Figures A1-1 for a complete listing of ids for Transport Layer users.
	Message starting address	address	IN	Location for the input message to be moved to
	Source GPC	gpc_t	OUT	GPC that sent the message. Refer to description of gpc_t given previously for IDENTIFY_TASK_LOCATION routine.
	Source user	lc_user_id	OUT	User that sent the message. Refer to Figures A1-1 and A1-2 for a complete listing of ids for Transport Layer users.
	Message priority	msg_priority_t	OUT	The user-assigned priority which ranges from 1 (lowest) to 100 (highest).
	Input-available flag	boolean	OUT	Flag that indicates whether or not there was a message for the caller. If there was, it has been moved to the address specified by the caller. (In the case of tasks scheduled specifically when there is input, this parameter is redundant.)

Figure B-6. Parameters Required for the Interface Routines (cont.)

DATA TYPE	PACKAGE	DATA TYPE	PACKAGE
a_Event	LSS_EVENT_CNTL		
address	SYSTEM		
gpc_t	ICCS_USER_DATA_TYPES		
lc_user_id	ICCS_SYSTEM_USER_IDS, ICCS_APPLIC_USER_IDS		
msg_priority_t	ICCS_USER_DATA_TYPES		
output_error_t	ICCS_USER_DATA_TYPES		
output_status_t	ICCS_USER_DATA_TYPES		
processor_id	ICCS_USER_DATA_TYPES		
task_id	LSS_TASK_IDS		

Figure B-7. Packages Defining Required Data Types

B.3 Example

Figure B-8 is a partial listing of code from a Sensor Processing application that was written for demonstration purposes. This listing shows the calls to the various Transport Layer interface routines. Because this task needs to execute periodically, whether or not it has received any input, it polls for incoming messages.

Figure B-9 shows how a task might schedule itself to execute only when there are input messages waiting for it.

```
------------------------------------------------------------------------
--              SENSOR PROCESSING Application
--
--  Simulates reading various sensors.
--  Sends the sensor data to a Central Command & Processing application
------------------------------------------------------------------------

with ICCS_APPLIC_USER_IDS, ICCS_USER_DATA_TYPES, ICCS_USER_SERVICES_APP;
use  ICCS_APPLIC_USER_IDS, ICCS_USER_DATA_TYPES, ICCS_USER_SERVICES_APP;

with LSS_TASK_IDS;

with ICDEMO_APPLIC_DATA;
use  ICDEMO_APPLIC_DATA;


package ICDEMO_SENSOR_PROC is

------------------------------------------------------------------------
--  Task definition
------------------------------------------------------------------------

task type SENSOR_PROC_t is
    pragma priority (45);
end SENSOR_PROC_t;

SENSOR_PROCESSING : SENSOR_PROC_t;

MY_ID : constant ic_user_id := SENSOR_PROC;


function Get_Id is new LSS_Task_Ids.Id_Of (SENSOR_PROC_t);

Sensor_Proc_Id : LSS_Task_Ids.Task_Id := Get_Id (SENSOR_PROCESSING);

b : boolean := IDENTIFY_TASK_LOCATION (MY_ID,
                                       SENSOR_PROC_SITE,
                                       CP,
                                       Sensor_Proc_Id);

end ICDEMO_SENSOR_PROC;
```

Figure B-8. Example: Sensor Processing Application

```
with LSS_EVENT_CNTL, UNCHECKED_CONVERSION;

with LSS_SCHEDULER;
use  LSS_SCHEDULER;


package body ICDEMO_SENSOR_PROC is

                            --  output to CCP task; i.e.,
                            --  valid sensor data
valid_sensor_data : valid_sensor_data_t;

                            -- input from CCP task, i.e.,
                            -- validation limits or sensor mode commands
sensor_info : sensor_info_t;
validation_limits : sensor_info_t;
sensor_mode_cmds  : sensor_info_t;

                            -- output to Flight Safety, i.e.,
                            -- isds signals
isds_signals : isds_signals_t;


max_output : constant integer := 70;   -- max size of output record
-------------------------------------------------------------------------
--  SENSOR PROCESSING task
-------------------------------------------------------------------------

task body SENSOR_PROC_t is


dummy_input_event : LSS_Event_Cntl.a_Event;

source_gpc : gpc_t;
source_task : ic_user_id;
input_to_process : boolean;
in_msg_prio : msg_priority_t;
error_code : output_error_t;
execution_count : integer;

begin

-------------------------------------------------------------------------
--  INITIALIZATION required for ICCS users
-------------------------------------------------------------------------

   Input_Setup (MY_ID,            -- allocate input buffers
                6,
                sensor_info'size/8,
                false,
                dummy_input_event);


   Output_Setup (MY_ID,           -- allocate output buffers
                 4,
                 max_output);


   --   Other initialization
   --
   --
   --
```

Figure B-8. Example: Sensor Processing Application (cont.)

```
-------------------------------------------------------------------------
--  MAIN TASK LOOP
-------------------------------------------------------------------------

   loop

      WAIT_FOR_SCHEDULE;

      --  First see if I have any input (validation limits or
      --  sensor mode commands) from the CCP task

      Get_Input (MY_ID,
                 sensor_info'address,
                 source_gpc,
                 source_task,
                 in_msg_prio,
                 input_to_process);

      while input_to_process loop        -- get all input messages

           if source_task = CCP then     -- input from Central Command & Processing ...
                --
                --

          else                          -- input I wasn't expecting
               null;
          end if;

          Get_Input (MY_ID,   -- see if there's more input
                     sensor_info'address,
                     source_gpc,
                     source_task,
                     in_msg_prio,
                     input_to_process);

      end loop;   -- while input_to_process ...


      --  Read the various sensors here and generate valid sensor
      --  data record for the Central Command/Processing task

      --
      --
      --

      Send_Output (My_Id,                  -- send the data
                   valid_sensor_data'address,
                   CCP_SITE,
                   CCP,
                   valid_sensor_data'size/8,
                   1,      -- msg priority
                   0,      -- user msg id
                   false,  -- no delayed error checking
                   error_code);

       if error_code /= NO_ERRORS then
            null;       -- Do necessary error handling here
       end if;

       --  Continue with processing
       --
       --
       --
```

Figure B-8. Example: Sensor Processing Application (cont.)

```
-------------------------------------------------------------------------
--  RANGE_SAFETY task
-------------------------------------------------------------------------

task body RANGE_SAFETY_t is

my_input_event : LSS_Event_Cntl.a_Event;

source_gpc  : gpc_t;
source_task : ic_user_id;

input_to_process : boolean;
in_msg_prio : msg_priority_t;

function gen_to_isds is new UNCHECKED_CONVERSION
                                (gen_purpose_p, isds_signals_p);
function gen_to_de   is new UNCHECKED_CONVERSION
                                (gen_purpose_p, flight_destruct_enable_p);
function addr_to_gen is new UNCHECKED_CONVERSION
                                (System.address, gen_purpose_p);


begin

-------------------------------------------------------------------------
--  INITIALIZATION required for ICCS users
-------------------------------------------------------------------------

   Input_Setup (MY_ID,                  -- allocate input buffers
                4,
                gen_purpose_rec'size/8,
                true,
                my_input_event);

   Schedule (Range_Safety_Id,         -- schedule myself to run when there's
             false,                    -- input
             Same_Priority,
             (On_Event_Set, my_input_event, true),
             No_Repetition,
             No_Completion);


-------------------------------------------------------------------------
--  MAIN TASK LOOP
-------------------------------------------------------------------------

   loop

      WAIT_FOR_SCHEDULE;

      Get_Input (MY_ID,
                 gen_purpose_rec'address,
                 source_gpc,
                 source_task,
                 in_msg_prio,
                 input_to_process);

      -- Do processing ...
      --
      --
      --
```

Figure B-9. Example: Task Scheduled by Arrival of Input

APPENDIX C. ICIS HARDWARE IMPLEMENTATION

A block diagram of the ICIS is shown in Figure C-1. The ICIS is programmed by an FTP which has access to the dual port memory and hardware registers that are associated with the ICIS. The ICIS utilizes a time shared 8K x 8 x 3 memory for program and input/output buffer storage. More specifically, there are three 8K x 8 memory devices in each ICIS; one device is associated with each of the three redundant network layers. This memory can be alternately accessed by the FTP and the ICIS microsequencers. An overview of the major logic blocks of the ICIS is given in Figure C-1. This section describes the instructions and registers available to ICIS users.

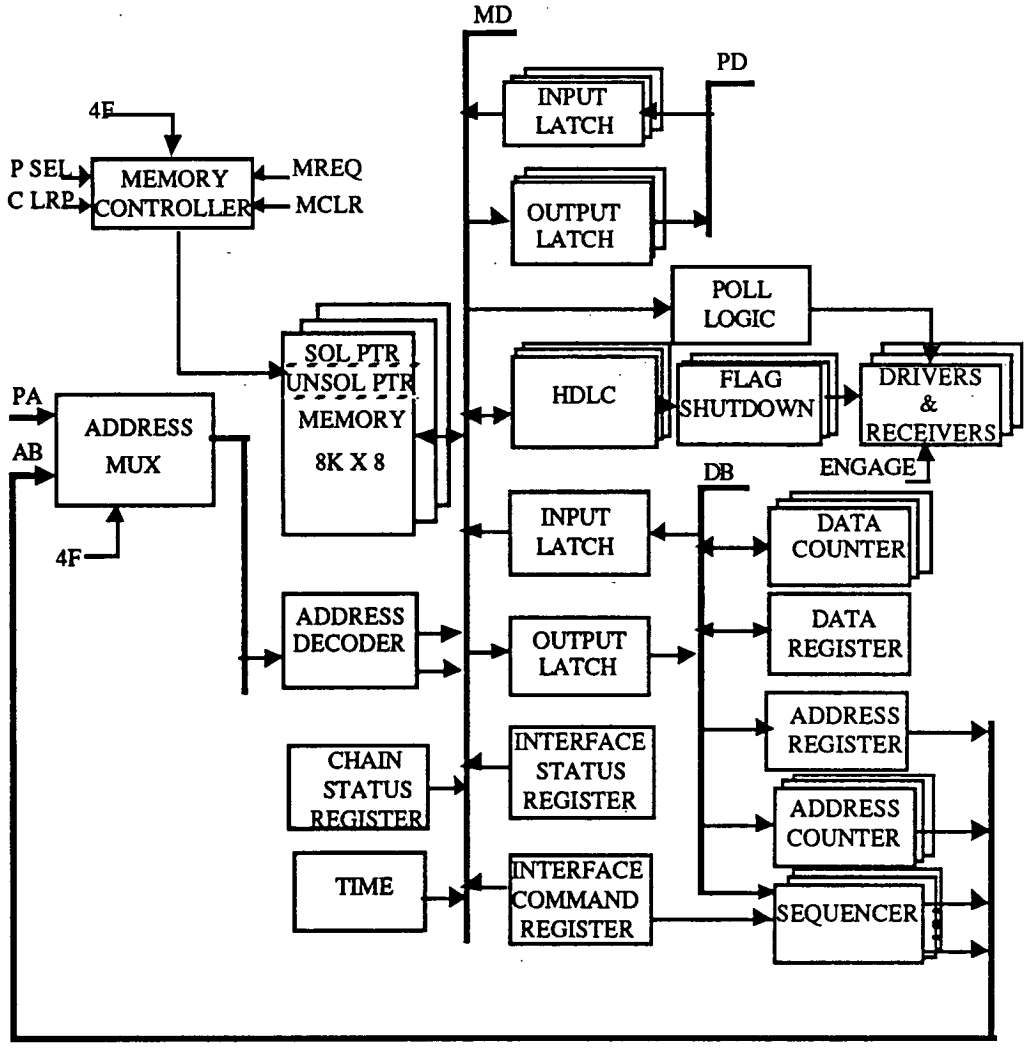


Figure C-1. ICIS Logic Blocks

ICIS Logic Blocks

MEMORY CONTROLLER - The Memory Controller arbitrates between memory accesses from the FTP and the ICIS. The memory is time shared via the processor signal $4F_{16}$. When $4F_{16}$ is high, the CP or IOP can access the memory and when it is low, the ICIS can access it. The memory controller generates chip select, read-write, and output enable at the appropriate times.

ADDRESS MULTIPLEXER - The Address Multiplexer selects between the FTP and ICIS address buses. The output of the multiplexer is the memory address bus (MA). When $4F_{16}$ is high, the processor address bus is connected to memory and when it is low, the ICIS memory bus is connected to the memory.

MEMORY - The ICIS memory consists of three byte-addressable memory devices each containing 8192 bytes. There is one memory device for each of the three network layers. The redundancy of the memory devices provides separate storage for each of the three independent data streams (one stream generated per the network layer). Note that the address and data buses for the memory devices are triplicated as well as all of the latches and multiplexers used to control accesses to the memory devices. This memory is also used to store the instruction chains and output packets. The first two bytes of memory are used for the solicited chain pointer and the second two bytes are reserved for the unsolicited chain pointer.

FTP INPUT LATCH - The FTP input latch is a buffer driver used to transfer data from the FTP data bus (PD) to the memory data bus (MD).

FTP OUTPUT LATCH - The FTP output latch is a buffer driver used to transfer data from the memory bus (MD) to the FTP data bus (PD).

ICIS INPUT LATCH - The ICIS input latch is a buffer driver used to transfer data from the internal ICIS data bus (DB) to the memory data bus (MD).

ICIS OUTPUT LATCH - The ICIS output latch is a buffer driver used to transfer data from the memory bus (MD) to the internal ICIS data bus (DB).

ADDRESS DECODER - The Address Decoder decodes the individual hardware registers which are located in the memory space between 10_{16} and $1F_{16}$. The addresses and details of the hardware registers are described in a following section.

INTERFACE COMMAND REGISTER - The Interface Command Register is a write only register that contains the command mode. See the following section on the details of the ICIS registers.

SEQUENCER - The Sequencer is the main control element of the ICIS. This logic block can be further subdivided into a main sequencer associated with the L network layer and two slave sequencers associated with the M and N layers. When started the main sequencer fetches the instructions from the L layer memory, stores them internally, and decodes and executes the microcycles by generating the appropriate control signals. The slave sequencers are only released for independent operation during the execution of an INPUT instruction. In this situation, they are responsible for managing the transfer of incoming data from the HDLC device to the local layer-specific memory devices.

CHAIN STATUS REGISTER - The Chain Status Register is a read only register that contains the chain and contention logic status within the ICIS. See the following section on the details of the ICIS registers.

INTERFACE STATUS REGISTER - The Interface Status Register is a read only register that contains the status of the ICIS. See the following section on the details of the ICIS registers.

ADDRESS COUNTER - The Address Counter stores the current memory address that the ICIS is using. It points to the chain instructions. During an INPUT instruction, it points to the location where the incoming data byte is to be stored. During an OUTPUT instruction, it points to the byte to be next sent. It is loaded during instruction fetches and incremented during the instruction microcycles. The address counter is triplicated; one exists for each layer.

ADDRESS REGISTER - The Address Register contains the fixed addresses used in the instructions. During an INPUT instruction, it contains the address of the HDLC input register. During an OUTPUT instruction, it contains the address of the HDLC transmitter holding register.

DATA COUNTER - The Data Counter contains any data that is incremented during an instruction. During an INPUT instruction, it accumulates the byte count of the incoming data. During an OUTPUT instruction, it counts the number of bytes transmitted. In the latter case, after the output data has been sent, it signals the sequencer to terminate the instruction. The data counter is triplicated.

DATA REGISTER - The Data Register is used to temporarily store data within an instruction. During an INPUT instruction, it holds the incoming byte from the HDLC receiver register until a memory cycle can be performed to store it. During an OUTPUT instruction, it holds the next byte to be outputted until the HDLC transmit holding register requests it. The data register is triplicated.

HDLC - The HDLC device contains independent transmitter and receiver sections. The HDLC transmitter section receives the data bytes, appends opening and closing flags, and encodes and transmits the data. The receiver section searches the data stream for an opening flag. When it detects one, it synchronizes itself with the data fields and decodes the data stream into bytes for storage. In both modes, the device generates the handshaking signals necessary to control the interface. The HDLC device is triplicated; one device interfaces with each individual network layer.

FLAG SHUTDOWN - The flag shutdown logic guarantees that the external IC network transmissions lines are always left in the same state. The ICIS uses the same IC network lines to communicate and poll. In order to be able to perform both functions on the same lines, all operations must leave the lines in a known state. The HDLC protocol allows the signalling lines to be left in either state, and in fact the device used to generate the HDLC protocol does leave the line in either state depending upon the data content of the message. The ICIS contains additional logic, which upon sensing the end of a message, utilizes the closing flags to turn off output with the line in a low state without generating any extraneous data. When the next output message is started, the first flags are used to turn the logic back on to the state that the HDLC device attempted to leave the line. Again this is done without generating any extraneous bits. The polling logic is designed so as to always end with the line low. The flag shutdown logic is triplicated.

DRIVERS and RECEIVERS - These drivers and receivers allow the ICIS to interface to the IC network. The drivers are enabled by an engage line from the FTP. The receivers are always enabled but the input is controlled by the HDLC device. All drivers and receivers are triplicated.

POLL LOGIC - The poll logic allows the ICIS to contend with other ICISes to gain control of the IC network. When enabled, the poll logic monitors the IC network waiting for a quiet time and then starts a poll. If it wins, it starts a solicited chain. Alternatively, if it loses, it waits for the next poll or quiet time and tries again.

If the ICIS is to contend for the network, the bits in the Interface Command Register must be set to "execute, poll, and execute unsolicited" mode. The logic will start the chain that is identified by the unsolicited pointer while it simultaneously primes the polling logic.

Note that the instruction distinguished by the unsolicited pointer is not the "current" instruction. For example, in the case where the current instruction is an INPUT which is waiting for unsolicited input, this instruction will be preempted by this polling and transition to solicited mode operation. A subsequent transition back to unsolicited mode will cause the fetch of the instruction following the INPUT instruction. Further,

the input buffers associated with the INPUT instruction will have header values as initialized at the start of the INPUT instruction (i.e., a byte count of zero).

The polling logic waits for either a poll to begin or the bus to go quiet for 512 microseconds. When either occurs, the logic asserts a start bit for 48 microseconds on all enabled network layers. This gives all other ICISes time to recognize the start of a poll and join if required. At the end of the poll bit, the logic compares the state of its input lines from the individual layers with the state of its output line. (Note that the state of the redundant input lines are either voted in Triplex mode or OR'd in all other modes). If another ICIS is joining the poll, the input line will be high and the ICIS must continue to poll. It next asserts the redundancy encoding priority bits, one at a time for 28 microseconds, and then asserts the remaining 9 priority bits. At the end of each 28 microsecond period, it compares its output to what it perceives on the bus. If what it receives is the same as what it transmits, it must continue to the next bit because no decision can be made. If it receives a zero when it is transmits a one, then the ICIS has won because it has a higher value than all others that are contending. If it receives a one while it is transmits a zero, then it has lost because it has a lower value than at least one other contender. As a result, it will stop transmitting and wait for another poll to begin. When the ICIS decides that it has won, it will abort the unsolicited chain and perform a context switch to the solicited chain and subsequently execute it.

If the IOL bit is set in the Interface Command Register, the three variable priority bits are incremented after each loss of a poll sequence (until they reach the maximum value of 7). They will remain at the maximum value until altered by an FTP program or an ICIS chain. If an ICIS detects a data bit while it is polling, it will terminate the poll and set the error bit Poll TX Fail in the Chain Status Register.

TIME DRIVER - The time driver allows the chain to read a time byte that appears on the shared bus.

MONITOR INTERLOCK ENGAGE - The AIPS FTPs generate a voted engage signal which is used to enable external functions. In a faulty FTP this signal will not be asserted. The ICIS uses this signal to enable its bus driver, a device that connects it to the IC network. Therefore, a faulty FTP and/or faulty ICIS can be disconnected and prevented from bring down the IC network.

ICIS Instruction Formats

The ICIS can execute a limited number of instructions. The following paragraphs detail the form and function of the ICIS instructions.

NOP (0000 0000) - This instruction updates the chain pointer to the address of the next sequential instruction. At the end of the NOP, the ICIS fetches that instruction.

BRANCH (2000 dddd) - This instruction jumps to the instruction contained at location 'dddd' and executes it. The Chain Pointer will be updated to point to the next instruction (dddd+4).

MOVE (40ss dddd) - This instruction will move a byte, located at any location 'ss' within the first 256 bytes of ICIS memory, to the location specified by 'dddd'. MOVE can be used to store the current value of a hardware register or store a preset value into a register.

MOVE IMMEDIATE (60xx dddd) - This instruction allows a constant, xx, to be stored into the destination dddd.

INPUT (801B dddd) - This instruction will store incoming HDLC bytes in the buffer area specified by 'dddd'. At the start of execution of this instruction, the byte reserved for the input byte count is set to zero and the current value of the contention status is stored within the data buffer. As bytes are received, they are stored at the specified buffer locations and an internal byte count is incremented. The incoming data streams for the redundant network layers are buffered independently in the redundant memories and independent byte counts are maintained for each layer.

Completion of the INPUT instruction is determined by the incoming layer activity. A valid HDLC packet always ends with a closing flag. Logic is provided to deal with the possibility that the incoming packets are skewed in time across the three layers. Once a closing flag is seen on any one of the three layers, a 12 microsecond delay in the termination of the INPUT instruction is provided if either of the other two layers indicate data activity (i.e., they are not idle). Instruction termination causes the ICIS to then store the byte count, HDLC status registers, and the TIME byte within the incoming packet buffer area. The INPUT instruction has now completed and the next sequential instruction is fetched and executed. The maximum number of data bytes that a single instruction can store is 122. If the INPUT contains more than 122 data bytes, data will be lost. However, the buffer never exceeds the 128 bytes allotted to it. Additionally, the byte count, which includes the status bytes, never exceeds 128 bytes. Furthermore, this instruction ends if the time allotted for response is exceeded (the value programmed into the timer is reached before a data byte being received).

However, in this situation, none of the status information (HDLC IR & SR registers, time and byte count) is saved.

An incoming data packet always has the following format:

Byte count
HDLC IR register
HDLC SR register
TIME byte
contents of Chain Status Register
data (first byte)
.
.
data (last byte received)

OUTPUT (E01C ssss) - This instruction will transmit the bytes specified in the buffer starting at location 'ssss + 1'. The first byte at location 'ssss' contains the value of the expression, 80_{16} - NB, where NB is the number of bytes to be transmitted. This instruction terminates when all bytes have been transmitted. The format of the output buffer is as follows:

Byte Count (80 16- NB)
data
.
.
last data byte

ICIS Memory Map

The reserved memory locations in the dual port memory of the ICIS are described below. Addresses 10_{16} - $1F_{16}$ are hardware registers; however, they are addressed in the same manner as RAM locations. All memory addresses, including the hardware registers, are accessible by the CP and IOP.

ADDRESS		FUNCTION
0	R/W	Solicited Chain Pointer - High Byte (RAM)
1	R/W	Solicited Chain Pointer - Low Byte (RAM)
2	R/W	Unsolic. Chain Pointer - High Byte (RAM)
3	R/W	Unsolic. Chain Pointer - Low Byte (RAM)

10	R	Chain Status Register
11	W	Interface Command Register
11	R	Interface Status Register
12	W	Timer Limit Register
13	W	Poll ID Register - 6 bit polling address
13	R	L&M Network States
14	W	Poll Prio Register-3 bit prio & polling level
14	R	Poller and N States
15	R	Time
15	W	"Location 15 Register"
16		Reserved
17		Reserved
18	R/W	HDLC Control Register 1 (CR1)
19	R/W	HDLC Control Register 2 (CR2)
1A	R/W	HDLC Control Register 3 (CR3)
1B	R	HDLC Receiver Holding Register (RHR)
1B	W	Address Register (AR)
1C	R	HDLC Interrupt Register (IR)
1C	W	Transmit Holding Register(THR)
1D	R	HDLC Status Register (SR)
1E		Reserved
1F		Reserved

With the exception of the addresses specified above, the rest of the ICIS's dual port memory can be used for any desired function. However, it should be noted that the MOVE instruction can only use the first 256 addresses for the source byte.

A description of the hardware registers and their use is presented in the following paragraphs.

SOLICITED CHAIN POINTER (ADDR = 00 & 01) - The ICIS can execute two types of chains, solicited and unsolicited. Solicited chains are defined as command/response chains and are meant to be executed when the FTP has control of the network. Unsolicited chains are defined as those that are performed when the FTP does not have control of the network but when it must accept all frames addressed to it. On the IC network, unsolicited chains are executed whenever the FTP does not possess the network, including while waiting for a poll to be won.

The Solicited Chain Pointer is used by the ICIS to indicate where the next instruction of a solicited chain is located. When a new chain is to be started, this location is loaded with the address of the first instruction to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The FTP can read this location at

any time. However, since the ICIS writes the locations a byte at a time and the FTP reads them as words, the value read by the FTP may be incorrect if a chain is executing. Therefore, the FTP should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the ICIS is not concurrently modifying them.

UNSOLICITED CHAIN POINTER (ADDR = 02 & 03) - The Unsolicited Chain Pointer is used by the ICIS to indicate where the next instruction of an unsolicited chain is located. When a new chain is to be started, this location is loaded with the address of the first instruction of the unsolicited chain to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The FTP can read this location at any time. However, since the ICIS writes the locations a byte at a time and the FTP reads them as words, the value read by the FTP may be incorrect if a chain is executing. Consequently, the FTP should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the ICIS is not simultaneously modifying them. Unsolicited chains are identical to solicited chains and can execute any mix of instructions.

CHAIN STATUS REGISTER (ADDR = 10) - This register contains status of both the chain and the contention logic.

7	6	5	4	3	2	1	0
Chain Complete	Contention State 1	Contention State 2	Possession Default	Poll TX Fail	Data TX Fail	Any Rcv Fail	Any Rcv Good

CSR

CHAIN COMPLETE (bit 7) - This bit is set whenever the current chain has completed. "Chain complete" is defined as an ICIS transition from solicited to unsolicited mode without the POLL bit in the Interface Command Register set. The Chain Complete bit is reset whenever the POLL bit is changed to a one or the ICIS transitions from the unsolicited to the solicited mode.

CONTENTION STATE (bits 6 and 5) - This is the current state of the poll logic. The following are the possible states that can be indicated:

INACTIVE, BUS RELEASED (00) Both bits are zero whenever the ICIS is not attempting to gain control of the network.

WAIT (01) This ICIS has been instructed to acquire the network; however, no POLL has completed since the request occurred.

ATTEMPTED (10) This ICIS has entered and lost at least one POLL sequence since being commanded to acquire the network.
POSSESSES (11) This ICIS currently has possession of the network.

POSSESSION DEFAULT (bit 4) - Indicates that the ICIS possesses the network and detected an incoming POLL length bit on the network. This bit is reset whenever the POLL bit in the Interface Command Register is set to zero.

POLL TX FAIL (bit 3) - Indicates that a data length bit was detected during a Poll Sequence. This bit is reset whenever the POLL bit in the Interface Command Register is set to zero.

DATA TX FAIL (bit 2) - Indicates that a data bit was detected at the receiver during a command frame transmission. The chain will continue to completion. This bit is reset whenever the POLL bit in the Interface Command Register is set to zero. This bit can only be set during a network possession.

ANY RCV FAIL (bit 1) - Indicates that at least one response frame has been received with a protocol error in it. It is reset whenever a new poll begins or the ICIS transitions from the unsolicited to the solicited mode.

ANY RCV GOOD (bit 0) - Indicates that at least one response frame has been received without a protocol error. It is reset whenever a new poll begins or the ICIS transitions from the unsolicited to the solicited mode.

INTERFACE COMMAND REGISTER (Write Only) (ADDR = 11) - This register contains the necessary control bits to operate the ICIS sequencer.

7	6	5	4	3	2	1	0
Execute Chain	x	Stop Immed.	Poll	Spoll	Exec. Unsol. Chain	x	Incr. on Lose

EXECUTE CHAIN (bit 7) - When only the execute chain bit is set, the ICIS is instructed to fetch and execute the instructions which start at the address stored in the Solicited Chain Pointer. The chain will start even if a poll was neither started nor won. If a poll is to be won before the chain is to be executed, then bits 7, 4 and 2 must all be set to a one. The hardware will then start the polling logic, start an unsolicited chain pointed to by the Unsolicited Chain Pointer, and when a poll is won, automatically start the chain pointed to by the solicited chain pointer.

STOP IMM (bit 5) - When the Stop Immediately bit is set to a one, the hardware turns off the ICIS. Whatever function the ICIS is performing is terminated. This allows the FTP to stop the ICIS hardware if the ICIS is caught in a loop or otherwise malfunctioning.

POLL (bit 4) - Whenever the POLL bit is set to a one, the logic attempts to gain control of the network by joining the next possible poll sequence. At the end of a chain, this bit must be reset.

SPOLL (bit 3) - Whenever the SPOLL bit is set to a one, the hardware will immediately start to poll. The hardware will not wait for the start of a new poll from another site or an idle condition on the network. At the end of a chain, this bit must be reset.

EXECUTE UNSOL CHAIN (bit 2) - This bit is only recognized by the hardware when set in conjunction with the execute chain bit, bit 7. If bits 7 and 2 are both set to a one, the hardware will execute the chain starting at the location pointed to by the Unsolicited Chain Pointer. If an FTP desires to first gain control of a network, it sets bits 7, 4 and 2 to a one and all others to a zero. The hardware will then enable the polling logic, start the unsolicited chain at the location pointed to by the Unsolicited Chain Pointer (usually an input instruction) and when a poll is won, automatically start the chain at the location pointed to by the Solicited Chain Pointer.

INCREMENT ON LOSE (bit 0) - This bit controls whether or not the first 3 poll priority bits (the polling bits used for network contention) are incremented automatically after each round of polling that is lost by the ICIS. The intention is to allow a sites of equal redundancy to fairly arbitrate with each other such that no one site always holds off another site during contentions for the network. If this bit is not set, then no autoincrementing is enacted.

The following are valid commands used to control the ICIS:

START CHAIN WITH POLL = 94
START CHAIN WITHOUT POLL = 80
END CHAIN = 84
STOP CHAIN = 20

The END CHAIN command transitions the ICIS from solicited to unsolicited mode. The STOP CHAIN command turns the ICIS off.

INTERFACE STATUS REGISTER (Read Only) (ADDR = 11) - This register contains status regarding the reception of unsolicited input and the stuck status of the individual network layers.

7	6	5	4	3	2	1	0
X	X	X	N STUCK	M STUCK	L STUCK	STUCK	UIR

ISR

UNSOLICITED INPUT RCVD--UIR(bit 0) - Set if any unsolicited input is received. This bit is set when the end of that message is detected and reset when this register is read.

STUCK (bit 1) - This bit is set to a one when any one of the three network monitors detects a stuck condition on a layer. The stuck condition is defined as a state in which the monitored layer signal has remained in a high position for more than 512 microseconds.

L STUCK (bit 2) - This bit is set to a one when the network monitor for the L layer detects a stuck condition.

M STUCK (bit 3) - This bit is set to a one when the network monitor for the M layer detects a stuck condition.

N STUCK (bit 4) - This bit is set to a one when the network monitor for the N layer detects a stuck condition.

TIMER LIMIT REGISTER (Write only) (ADDR = 12) - The timer limit register contains the current value to be used to time out an instruction. A non-zero value written to the timer limit register allows the timer to function. The timer is initialized at the beginning of each instruction and as each incoming data byte is detected. If an instruction does not complete or an incoming data byte is not detected in the programmed number of microseconds, the current instruction is terminated and the next sequential instruction started. A new value stored in the timer limit register will be accepted when the next instruction is started or the next incoming byte is accepted during an input instruction. The timer limit is the number of periods of the clock $2F_{16}$. $2F_{16}$ has a period of approximately 2 microseconds. The timer has a range of 2 to 512 microseconds.

POLL ID REGISTER (Write only) (ADDR = 13) - The Poll ID register contains the six (6) low order bits used in the polling procedure. These bits normally contain the address that this ICIS uses for polling. It can be written to by the FTP or by a MOVE instruction within the chain.

7	6	5	4	3	2	1	0
X	X	Bit 5 MSB	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 LSB

PPR

L&M NETWORK STATES (Read only) (ADDR = 13) - This register provides a window into the current state of layers L and M as reported by the network monitor logic that is associated with these layers. The encoding of the states of the layers is:

7 - Data
6 - Poll
5 - Wait
4 - Released
3 - Poll Detect
2 - Stuck
1 - Poll Deskew
0 - Idle

7	6	5	4	3	2	1	0
X	M State 2	M State 1	M State 0	X	L State 2	L State 1	L State 0

L&M States

M LAYER STATE (bits 6..4) - indicates current output of M layer's Network Monitor.

L LAYER STATE (bits 2..0) - indicates current output of L layer's Network Monitor.

POLL PRIORITY REGISTER (Write only) (ADDR = 14) - The Poll Priority Register provides the user interface into the network polling mechanism. This register has bits to enable/disable polling, to specify the redundancy level of this polling site, and to specify the initial 3 priority bits to be used in arbitrating for network possession.

7	6	5	4	3	2	1	0
X	Enable Poll	Triplex Poll	Duplex Poll	X	PRIO 2	PRIO 1	PRIO 0

PPR

ENABLE POLL (bit 6) - This bit is the "start" bit in the polling sequence (i.e., the first bit sourced on the network to indicate the site is polling). If this bit is not set, then the site will never start a poll and therefore should never obtain network possession.

TRIPLEX POLL (bit 5) - This bit is to be set only by triplex sites. It is the second poll bit sourced in a polling sequence. Setting this bit also causes the polling logic to perform a vote of the incoming polling data across channels as opposed to OR'ing this data.

DUPLEX POLL (bit 4) - This bit is to be set only by duplex sites. It is the third poll bit sourced in a polling sequence.

PRIORITY BITS (bits 2..0) - The three bits labeled PRIO are the initial priority of this ICIS and are used to arbitrate among network sites of equal redundancy levels. If the arbitration is not resolved after using these three bits, then the polling continues with the six lower order bits in the Poll ID Register. If the IOL bit is set in the Interface Control Register, the three PRIO bits will automatically increment after each poll sequence loss until they contain all ones at which time incrementing is inhibited. This maximum priority is held until the register is reloaded.

POLL STATE AND N MONITOR (Read only) (ADDR = 14) - This register contains a combination of information related to the current state of the Polling State Machine and to the current layer state as reported by the N layer's network monitor. In addition, there is a single bit of information which reports on whether the variable poll priority values have auto-incremented to their maximum value.

7	6	5	4	3	2	1	0
Poll State 3	Poll State 2	Poll State 1	Poll State 0	Max Priority	N State 2	N State 1	N State 0

Poller and N States

POLL STATE (bits 7..4) - indicates the current state of the Polling State Machine:

15 - Not Contending
14 - Network Possession
13 - Waiting to begin Poll
12 - Network Release Quasi-Stable
11 - Start Poll Sequence First 12 microseconds
10 - Join Poll Sequence
9 - not used
8 - Aborted from Poll
7 - not used
6 - Aborted from Network Possession
5 - not used
4 - not used
3 - Start Poll Sequence Second 12 microseconds
2 - not used
1 - not used
0 - Lost Last Polling Sequence

MAX PRIORITY (bit 3) - when set, the MAX PRIORITY bit indicates that the three poll priority bits, the PRIO bits in the Poll Priority Register, have been incremented to the maximum value of 7.

N LAYER STATE (bits 2..0) - indicates current output of N layer's Network Monitor (see encodings for the L&M States Register).

TIME (read only) (ADDR = 15) - This byte contains a value that is slaved to the system timer. It is incremented by a 66 microsecond clock and is capable of measuring 16.830 milliseconds. It can be read by the FTP or by a MOVE instruction in the chain. It is automatically appended to all incoming frames that complete in a valid manner.

LOCATION 15 REGISTER (Write only) (ADDR = 15) - This register contains various controls bits associated with the polling mechanism and the enabling of outputs onto the network layers.

7	6	5	4	3	2	1	0
M1	M0	Poll Enable L	Poll Enable M	Poll Enable N	Output Enable L	Output Enable M	Output Enable N

LOCATION 15 REGISTER

M1 & M0 (bits 7..6) - These two bits control the "masking" operations performed by the voters associated with the exchange of ICIS polling states among the redundant ICISes. The encoding of these bits are:

00 - Simplex
01 - Duplex using neighbor on Right
10 - Duplex using neighbor on Left
11 - Triplex

POLL ENABLES (bits 5..3) - These bits determine whether the input from a particular layer is included in the polling operation. A set bit will cause the associated layer's input to be disabled with respect to the polling logic (i.e., the poll bit is considered to be a 0). A reset bit will allow the actual incoming poll data to be passed to the polling logic.

OUTPUT ENABLES (bits 2..0) - These bits determine whether any data (both HDLC data and poll data) is to be sourced onto a particular network layer. A set bit enables output; a reset bit disables output.

CONTROL REGISTER #1 (CR1) (ADDR = 18) - Control Register 1 is used to specify the transmitter parameters and the transmitter and receiver enables. It can be loaded by an FTP or by a MOVE instruction in the chain. The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system.

NOTE: This register must always be loaded after CR2 and/or CR3. If CR2 and/or CR3 are ever changed, CR1 must again be reloaded after the change even if there are no changes being made to CR1.

7	6	5	4	3	2	1	0
ACT REC	ACT TRA N	TC1	TC0	TCL1	TCL0	DTR	MIS C

CR1

ACT REC (bit 7) - If the Activate Receiver bit is set to a ZERO (0), the receiver is enabled to accept a data stream. When it is set to a ONE (1), the receiver will ignore any frames on the network.

ACT TRAN (bit 6) - If the Activate Transmitter bit is set to a ZERO (0), the encoder and transmitter are enabled to output onto the network. When it is set to a ONE (1), the HDLC device will not transmit data.

TC1 and TC0 (bits 5 and 4) - The Transmit Command bits program the device into the requested mode. In AIPS, the OUTPUT instruction will function properly only in the data mode. These bits and the modes that they generate are as follows:

bit 5	bit 4	MODE	FUNCTION
1	1	data	Outputs the contents of the transmitter holding register
1	0	abort	Generates an abort message (not used on AIPS)
0	1	flag	Transmits one flag character (not used on AIPS)

0	0	FCS	Generates the two CRC bytes and a closing flag (not used on AIPS)

TCL1 and TCL0 (bits 3 and 2) - These bits control the number of bits per character from the transmitter. In AIPS, this has been defined as 8 bit bytes. The definition of these bits follows:

bit 3	bit 2	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

DTR (bit 1) - Data Terminal Ready is a modem signal that is not used in this design and should be programmed to a ONE (1).

MISC OUT (bit 0) - Miscellaneous Output is a control signal not implemented in this design and should be programmed to a ONE (1).

CONTROL REGISTER #2 (CR2) (ADDR = 19) - Control Register #2 specifies the receiver parameters and other control functions as defined below. It can be loaded by an FTP or by a MOVE instruction in the chain. The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system.

7	6	5	4	3	2	1	0
EXT CONT	ADDR COMP	EXT ADDR	RCL1	RCL0	LOOP	SELF TEST	AUTO FLAG

CR2

EXT CONT (bit 7) - This bit extends the HDLC control field. It is not used on AIPS and must be programmed to a ONE (1).

ADDR COMP (bit 6) - This bit enables the on-chip address comparator. If set to a ZERO (0), the first byte after the opening flag will be compared to the byte

stored in the AR register. If equal, the data bytes that follow will be received. If the address compare is enabled and the address does not compare, all following data bytes will be ignored. If the ADDR COMP bit is set to a ONE (1), then address comparison is not performed in the chip and all bytes between the opening and closing flag are presented to the interface. In AIPS, the ICIS does use the address compare function.

EXT ADDR (bit 5) - This bit extends the HDLC address field. It is not used on AIPS and must be programmed to a ONE (1).

RCL1 and RCL0 (bits 4 and 3) - These bits specify the receiver character length. In AIPS this has been defined as 8 bit characters. The definition of these bits is as follows:

bit 4	bit 3	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

LOOP (bit 2) - Specifies HDLC loop mode, a test function, that is not implemented in the ICIS. This bit should always be programmed to a ONE (1).

SELF TEST (bit 1) - A diagnostic mode that is not implemented in the ICIS. This bit should always be programmed to a ONE (1).

AUTO FLAG (bit 0) - When this bit is set to a ZERO (0) and the transmitter is enabled, the chip will issue constant flag characters between frames. The ICIS design utilizes this function and therefore must be set to a ZERO during an output instruction.

CONTROL REGISTER #3 (CR3) (ADDR = 1A) This register is used to control the number of residual bits in a transmission. It can be loaded by an FTP or by a MOVE instruction in the chain. The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system. The definitions of these bits are as follows:

7	6	5	4	3	2	1	0
X	X	X	X	X	TRES2	TRES1	TRES0

CR3

TRES 2 - 0 (bits 2, 1 and 0) - These bits define the number of residual bits to be sent as the last character in a transmission. Messages sent to and from a NODE must contain three (3) residual bits. The definition of these bits are as follows:

bit 2	bit 1	bit 0	RESIDUAL BITS/FRAME
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

RECEIVER HOLDING REGISTER (RHR) (ADDR = 1B) This read-only register contains the received bytes as they are decoded from the frame. When executing an INPUT instruction, the ICIS automatically reads this location and stores the received characters into the specified location in the dual port memory.

INTERRUPT REGISTER (IR) (ADDR = 1C) This read-only register contains status information on the state of the HDLC operation. It can be read by the FTP or with a MOVE instruction within a chain. Bits 7 through 3 will accumulate information such that if the IR is read after several operations, it will have the "OR" of all those frames. The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system. The definition of the bits within this register is as follows:

7	6	5	4	3	2	1	0
Reom NO Error	Reom with Error	Xmit No Error	Xmit with Urun	DISC	DRQI	DRQO	INTRQ

IR

REOM NO ERR (bit 7) - When equal to a ZERO, this bit indicates that the frame was received without errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame.

REOM WITH ERR (bit 6) - When equal to a ZERO, this bit indicates that the frame was received with errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame. The errors that are reported are: CRC, overrun, invalid frame, and aborted frame.

XMIT NO ERR (bit 5) - When equal to ZERO, this bit indicates that the transmitted frame had completed without underrun errors.

XMIT WITH URUN (bit 4) - When equal to ZERO, this bit indicates that the transmitted frame had extra bytes inserted by the chip because the data was not available to the transmitter in the allotted time.

DISC (bit 3) - This bit is used with modems and, in this system, it has no meaning.

DRQI (bit 2) - When set to a ZERO, this bit indicates that there is a byte available in the Receiver Holding Register (RHR). Reading the RHR sets this bit to a ONE. The hardware uses a buffered copy of this bit when storing bytes into dual port memory during an INPUT instruction.

DRQO (bit 1) - When set to a ZERO, this bit indicates that the Transmit Holding Register (THR) is empty and requires another character to prevent an underrun error. Storing a byte into the THR sets this bit to a ONE. The hardware uses a buffered copy of this bit during an OUTPUT instruction to read a byte from the dual port memory and store it into the THR.

INTRQ (bit 0) - This bit is set to a ZERO whenever at least one of the other bits in the IR register is set to a ZERO. This bit is set to a ONE whenever the IR is read. A buffered copy of this bit is used to terminate a normally completing input or output instruction.

ADDRESS REGISTER (AR) (ADDR =1B) - This write-only register contains the address that the chip is to use for comparison if on-chip address recognition is being used. If on-chip address detection is not used, the contents of this register will be ignored.

TRANSMIT HOLDING REGISTER (THR) (ADDR = 1C) - This write-only register holds the next data byte to be transmitted. The hardware loads a byte into this register during an OUTPUT instruction whenever DRQO is set.

STATUS REGISTER (SR) (ADDR = 1D) - This read-only register contains status information that, when used in conjunction with the contents of the Interrupt Register, define the cause of the error.

7	6	5	4	3	2	1	0
RI	CD	DSR	MISC. IN	RCVR IDLE	RRES2 /ERR	RRES1 /ERR	RRES0 /ERR

SR

RI (bit 7) - A modem signal not implemented in this interface.

CD (bit 6) - A modem signal not implemented in this interface.

DSR (bit 5) - A modem signal not implemented in this interface.

MISC IN (bit 4) - An input discrete not used in this interface.

RCVR IDLE (bit 3) - When set to a ZERO, the receiver is idle, i.e. a frame is not in process.

RRES2 /ERR (bit 2) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number representing the number of residual bits received (see encoding after RRES0/ERR description). If bit 6 in the Interrupt Register is set to a ZERO and this bit is set to ZERO, then an aborted or invalid frame was detected.

RRES1 /ERR (bit 1) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number representing the number of residual bits received (see encoding after RRES0/ERR description). If bit 6 in the Interrupt Register is set to a ZERO and this bit is set to ZERO, then an overrun error was detected. An overrun error indicates that a received byte was not removed from the Receiver Holding Register before the next byte was received. In an overrun condition, that first byte is lost.

RRES0 /ERR (bit 0) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see encoding below) representing the number of residual bits received. If bit 6 in the Interrupt Register is set to a ZERO and this bit is set to ZERO, then a CRC error was detected.

bit 2	bit 1	bit 0	RESIDUAL BITS/FRAME
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

APPENDIX D: PROPOSED MODIFICATIONS TO ICCS

This appendix describes modifications that should be made to the ICCS Transport Layer. The first eight modifications are functional improvements and the last modification is for performance improvement.

1. Use One Output Queue Rather than Two

Currently output messages from CP users go into one queue while those from IOP users go into another queue. Therefore messages are not necessarily sent by the GPC in the same order in which they were generated (although they are sent in the order in which they were generated by a particular task or by a particular processor). Using a common queue for both processors would ensure that messages are sent in the correct order, in addition to simplifying the code.

2. Specify Additional Parameter for Output Messages

An additional parameter to the Send_Output routine should be implemented. This parameter would be a boolean "hold flag" that indicates whether or not a message should be put on the pending list if it cannot be sent immediately (because a destination GPC is already receiving its maximum messages). In the case of a broadcast it would, (by extension), indicate that the message should or should not be sent if it can't be sent in broadcast mode.

A "NO" value for this parameter (indicating messages should not be held, i.e., not put on the pending list) would be appropriate for periodic tasks sending messages at high rates. It would also be appropriate for broadcasts where it is important that the message arrive at all GPCs at the same time (although it still would "arrive" at the local GPC at a different time). A "YES" value for this parameter would be appropriate for one-time messages that **must** be sent and for which some delay in transmission is tolerable.

3. Restrict Length of Broadcast Messages

Broadcast messages should not be allowed to be longer than 104 bytes (i.e., what will fit into one packet). Messages longer than this are not sent in broadcast mode after the first packet anyway, and such a restriction would allow some performance improvements to be made (see item 9). A user with a message longer than 104 bytes could either divide it into two messages and do two separate transmissions, or execute a loop to send it to each GPC separately.

4. Check Message Size Against Buffer Size

When messages are moved into the temporary input or output buffers, a check should be made to verify that the buffer is big enough to hold the message. Similarly when a user calls the GET_INPUT routine, he should also pass the size of the area that the message is to be moved into, so it can be verified that the message is not bigger than this area.

5. Code and Test Use of New Scheduler Option for Overruns

As described in Section 4, the Message Send-Receive task is started by an event, which can be set either by the ICIS RM task when there are input packets to process or by User Services when a user has a message to send. Every time the task is scheduled, however, it checks its entire list of possible things to be done and continues to check as long as it has handled a new message during any particular iteration. It is possible, however, that its event could be set after it made its last check but before it got to its WAIT_FOR_SCHEDULE point. The original version of the Ada run-time Scheduler would have marked this as an overrun and not scheduled the task when it did get to its WAIT_FOR_SCHEDULE.

The SCHEDULE routine has been modified to allow a new parameter which specifies whether or not a task is to be scheduled after an overrun occurs. The Message Send-Receive task needs to be changed to use this option; then it must be tested.

6. Schedule Message Send-Receive Task Periodically to Check for Timeouts

The Message Send-Receive task is started by an event, as described in 5 above. Along with processing input and output messages, it checks every so often to see if any messages have timed out, i.e., an output message has not received an ACK or a multi-packet input message has not received a MSG_CONT. If the task were not triggered by the event, however, i.e., no new messages were being generated, this task would not run and therefore would not check for timeouts. The task also needs to be scheduled on a periodic basis as well as on an event basis, yet without doing unnecessary context switching. The solution to this problem needs to be designed as well as coded and tested.

7. Validate Message Destination in User Services Rather than Message Send-Receive

When the Message Send-Receive task tries to send an output message, it first verifies that the destination task is identified in either the CP Task Location Table or the IOP Task Location Table. If it is not, it makes an entry in the IC Error Log and ignores the message. This verification should be done by the SEND_OUTPUT routine and an error indication returned to the caller.

8. Validate Control Information in Incoming Messages

The control information in incoming messages (i.e., destination task, source task, message length, etc.) should be validated before attempting to use it. Some types will cause exceptions when they contain invalid data, but others will not, e.g., message length which is defined as a short_integer.

9. Eliminate Message Status Block Data Structure

In order to improve performance, the information kept in this structure could instead be kept in the temporary input and output buffers. Keeping it in these buffers would be greatly simplified if broadcast messages were limited to 104 bytes (see item 3). Not having to dynamically allocate the Message Status Block, maintain a linked list, and deallocate the block would save several milliseconds and even more in the case of a broadcast message where a separate block is allocated for each receiving GPC. This would, of course, increase the amount of memory required for the buffers.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-187556		

4. Title and Subtitle	5. Report Date
Advanced Information Processing System: Inter-Computer Communication Services	September 1991
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Laura Burkhardt, Tom Masotto, J. Terry Sims, Roy Whittredge, and Linda Alger	
	10. Work Unit No. 506-46-21-56

9. Performing Organization Name and Address	11. Contract or Grant No.
The Charles Stark Draper Laboratory, Inc. Cambridge, MA 02139	NAS1-18565
	13. Type of Report and Period Covered

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Felix L. Pitts
Final Report

16. Abstract

The purpose of this report is to document the functional requirements and detailed specifications for the Inter-Computer Communication Services (ICCS) of the Advanced Information Processing System (AIPS). An introductory section is provided to outline the overall architecture and functional requirements of the AIPS system and to present an overview of the Inter-Computer Communication Services. Section 1.1 gives an overview of the AIPS architecture as well as a brief description of the AIPS inter-computer network architecture; Section 1.2 provides an introduction to the AIPS system software; Section 1.3 provides the guarantees of the Inter-Computer Communication Services; and Section 1.4 describes the Inter-Computer Communication Services as a seven-layered International Standards Organization (ISO) model. Sections 2 through 6 describe the Inter-Computer Communication Services functional requirements, functional design and detailed specifications. Each of these sections describes one of the "Layers" of the Inter-Computer Communication Services. Section 7 concludes with a summary of results and suggestions for future work in this area.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
AIPS Inter-Computer Communication Services AIPS Inter-Computer Network AIPS Inter-Computer Network Services Fault-Tolerant Network Distributed Processors	Unclassified - Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	278	

NASA FORM 1626 OCT 86